Minimizing Overprocessing Waste in Business Processes via Predictive Activity Ordering

Ilya Verenich[1,2], Marlon Dumas[2,1], Marcello La Rosa[1,3], Fabrizio Maria Maggi[2], and Chiara Di Francescomarino[4]

[1] Queensland University of Technology, Australia
{ilya.verenich,m.larosa}qut.edu.au
[2] University of Tartu, Estonia
{marlon.dumas,f.m.maggi}@ut.ee
[3] NICTA Queensland, Australia
[4] FBK-IRST, Trento, Italy
dfmchiara@fbk.eu

Abstract. Overprocessing waste occurs in a business process when effort is spent in a way that does not add value to the customer nor to the business. Previous studies have identified a recurrent overprocessing pattern in business processes with so-called "knockout checks", meaning activities that classify a case into "accepted" or "rejected", such that if the case is accepted it proceeds forward, while if rejected, it is cancelled and all work performed in the case is considered unnecessary. Thus, when a knockout check rejects a case, the effort spent in other (previous) checks becomes overprocessing waste. Traditional process redesign methods propose to order knockout checks according to their mean effort and rejection rate. This paper presents a more fine-grained approach where knockout checks are ordered at runtime based on predictive machine learning models. Experiments on two real-life processes show that this predictive approach outperforms traditional methods while incurring minimal runtime overhead.

Keywords: Process mining; process optimization; overprocessing waste

1 Introduction

Overprocessing is one of seven types of waste in lean manufacturing [1]. In a business process, overprocessing occurs when effort is spent in the performance of activities to an extent that does not add value to the customer nor to the business. Overprocessing waste results for example from unnecessary detail or accuracy in the performance of activities, inappropriate use of tools or methods in a way that leads to excess effort, or unnecessary or excessive verifications [2].

Previous studies in the field of business process optimization have identified a recurrent overprocessing pattern in business processes with so-called "knockout checks" [3,4]. A knockout check is an activity that classifies a case into "accepted" or "rejected", such that if the case is accepted it proceeds forward, while if rejected, all other checks are considered unnecessary and the case is either terminated or moved to a later stage in the process. When a knockout check rejects a case, the effort spent in other (previous) checks becomes overprocessing waste. This overprocessing waste pattern is common in application-to-approval processes, where an application goes through a number of checks aimed at classifying it into admissible or not, such as eligibility checks in a University admission process, liability checks in an insurance claims handling process, or credit worthiness checks in a loan origination process. Any of these checks may lead to an application or claim being declared ineligible, effectively making other checks irrelevant for the case in question.

A general strategy to minimize overprocessing due to the execution of unnecessary knockout checks is to first execute the check that is most likely to lead to a negative ("reject") outcome. If the outcome is indeed negative, there is no overprocessing. If on the other hand we execute first the checks that lead to positive outcomes and leave the one that leads to a negative outcome to the end, the overprocessing is maximal – all the checks with positive outcome were unnecessary. On the other hand, it also makes sense to execute the checks that require less effort first, and leave those requiring higher effort last, so that the latter are only executed when they are strictly necessary. These observations lead to a strategy where knockout checks are ordered according to two parameters: their likelihood of leading to a negative outcome and the required effort.

Existing process optimization heuristics [3,5] apply this strategy at design-time. Specifically, checks are ordered at design-time based on their rejection rate and mean effort. This approach achieves some overprocessing reduction, but does not take into account the specificities of each case. This paper proposes an approach that further reduces overprocessing by incorporating the above strategy into a predictive process monitoring method. Specifically, the likelihood of each check leading to a positive outcome and the effort required by each check are estimated at runtime based on the available case data and machine learning models built from historical execution data. The checks are then ordered at runtime for the case at hand according to the estimated parameters.

The rest of the paper is organized as follows. Section 2 gives a more detailed definition of knockout checks and discusses related work. Section 3 presents the proposed knockout check reordering approach. Next, Section 4 discusses an empirical evaluation of the proposed approach versus design-time alternatives based on two datasets related to a loan origination process and an environmental permit process. Finally, Section 5 draws conclusions and outlines future work.

2 Background and Related Work

This paper is concerned with optimizing the order in which a set of knockout checks are performed in order to minimize overprocessing. The starting point for this optimization is a *knockout section*, defined as a set of *independent binary knockout* checks. By independent we mean that the knockout checks in the section can be performed in any order. By binary we mean that each check classifies the case into two classes, hereby called "accepted" and "rejected". And by knockout we mean that if the check classifies a case as "rejected", the case jumps to a designated point in the process (called an *anchor*) regardless of the outcome of all other checks in the section. An anchor can be any point in the process execution either before or after the knockout section. In the rest of the paper, we assume that the anchor point is an end event of the process, meaning that a case completes with a negative outcome as soon as one of the checks in the knockout section fails.

For example, a loan application process in a peer-to-peer lending marketplace typically includes several knockout checks. Later in this paper we will examine one such process containing three checks: identity check; credit worthiness check; and verification of submitted documents. Any of these checks can lead to rejection of the loan, thus the three checks constitute a knockout section.

The order of execution of checks in a knockout section can impact on overprocessing waste. For example, in the above knockout section, if the identity check is completed first and succeeds and then the credit worthiness check is completed and leads to a rejection, then the identity check constitutes overprocessing, as it did not contribute to the outcome of the case. Had the credit worthiness check been completed first, the identity check would not have been necessary.

Van der Aalst [3] outlines a set of heuristics for redesigning knockout sections. These heuristics resequence the knockout checks according to the average processing time, rejection rate and setup time of each check. One heuristic is to execute the checks in descending order of rejection rate, meaning that the checks that are more likely to reject a case are executed first. A more refined heuristic is one where the checks are executed in descending order of the product of their rejection rate times their required effort (average processing time). In other words, checks are ordered according to the principle of "least effort to reject" – checks that require less effort and are more likely to reject the case come first. This idea is identified as a redesign best practice by Reijers et al. [5] and called the "knockout principle" by Lohrmann and Reichert [6].

Pourshahid et al. [7] study the impact of applying the knockout principle in a healthcare case study. They find that the knockout pattern in combination with two other process redesign patterns improve some of the process KPIs, such as average approval turnaround time and average cost per application. Niedermann et al. [8] in the context of their study on process optimization patterns introduce the "early knockout" pattern. The idea of this latter pattern is moving the whole knockout section to the earliest possible point.

All of the above optimization approaches resequence the knockout checks at design time. In contrast, in this paper we investigate the idea of ordering the checks at runtime based on the characteristics of the current case. Specifically, we seek to exploit knowledge extracted from historical execution traces in order to predict the outcome of the knockout checks and to order them based on these predictions. In this respect, the present work can be seen as an application of *predictive process monitoring*.

Predictive process monitoring is a branch of process mining that seeks to exploit event logs in order to predict how one or multiple ongoing cases of a business process will unfold up to their completion [9]. A predictive monitoring approach relies on machine learning models (e.g. classification models) trained on historical traces in order to make predictions at runtime for ongoing cases. Existing predictive process monitoring approaches can be classified based on the predicted output or on the type of information contained in the execution traces they take as input. In this respect, some approaches focus on the time perspective [10], others on the risk perspective [11]. Some of them take advantage only of a static snapshot of the data manipulated by the traces [9], while in others [12,13], traces are encoded as complex symbolic sequences, and hence the successive data values taken by each data attribute throughout the execution of a case are taken into account. This paper relies on the latter approach. The main difference between the present work and existing predictive monitoring approaches is that the goal is not to predict the outcome of the entire case, but rather to predict the outcome of individual activities in the case in order to re-sequence them.

The idea of using predictive monitoring to alter (or customize) a process at runtime is explored by Zeng et al. [14] in the specific context of an invoice-to-cash process. The authors train a machine learning model with historical payment behavior of customers, with the aim of predicting the outcome of a given invoice. This prediction is then used to customize the payment collection process in order to save time and maximize the chances of successfully cashing in the payment. In comparison, the proposal outlined in this paper is generally applicable to any knockout section and not tied to a specific application domain.

3 Approach

In this section we describe the proposed approach to resequencing knockout checks in order to minimize overprocessing. We first give an overview of the entire solution framework and then focus on the core parts of our approach.

3.1 Overview

Given a designated knockout section in a process, the goal of our approach is to determine how the checks in this section should be ordered at runtime in order to reduce overprocessing waste. Accordingly, our approach pre-supposes that any preexisting design-time ordering of the checks be relaxed, so that instead the checks can be ordered by a runtime component.

The runtime component responsible for ordering the checks in a knockout section relies on a predictive monitoring approach outlined in Figure 1. This approach exploits historical execution traces in order to train two machine learning models for each check in the knockout section: one to predict the probability of the check to reject a given case, and the second to predict the *expected processing time* of the check. The former is a classification model while the latter is a regression model.

To train these models, the traces of completed cases are first encoded as feature vectors and fed into conventional machine learning algorithms. The resulting models are then used at runtime by encoding the trace of an ongoing case as a feature vector and giving it as input to the models in order to estimate the *expected processing effort* of each allowed permutation of knockout checks and to select the one with the lowest expected effort. To validate the models, once the case has completed and the actual outcome of the checks is known, we compute the *actual processing effort* and compare it with the *minimum processing effort* required to either accept or knock out the case in question. The difference between the actual and the minimum effort is the overprocessing waste.

3.2 Estimation of Expected Processing Effort

As mentioned in the introduction, overprocessing results from the activities that add no value to the product or service. For example, if knockout activity rejects

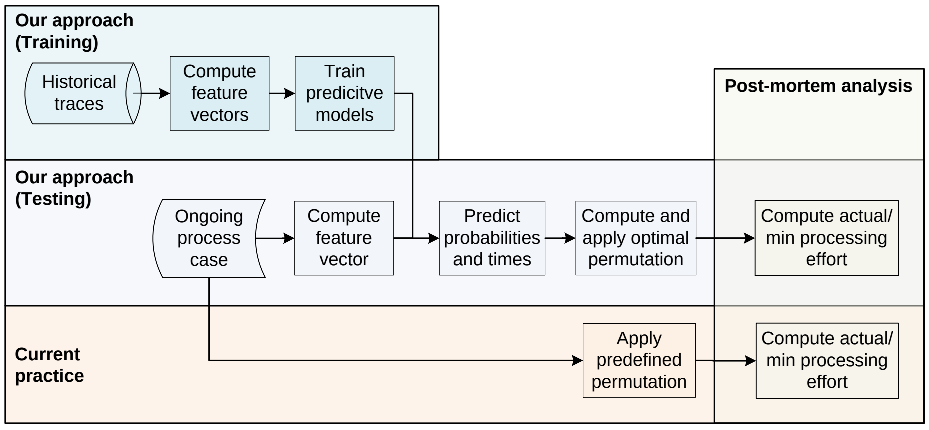


Fig. 1: Overview of the proposed approach.

a case, then the case is typically terminated and the effort spent on the previous activities becomes overprocessing waste. Consequently, to minimize the overprocessing, we are interested in determining such a permutation σ of activities that the case will be knocked out as early as possible. In the best case, the first executed activity will knock out the case; in the worst case, none of them will knock out the case. Furthermore, among all activities that could knockout the case, the one with lowest effort represents the minimal possible processing effort W_{min} for a particular case to pass the knockout section. If none of the activities knocks out the case, there is no overprocessing.

Since the minimal possible processing effort is constant for a particular process case, minimizing overprocessing of a knockout section is essentially equivalent to minimizing overall processing effort W_σ, which is dependent on the actual number of performed activities M in the knockout section:

$$W_\sigma = \sum_{i=1}^{M} w_i = \sum_{i=1}^{M} T_i R_i, \quad 1 \leq M \leq N \tag{1}$$

where w_i is the effort of an individual activity, T_i is its expected processing time and R_i is the cost of a resource that performs the activity per unit of time, which is assumed constant and known.

At least one activity needs to be performed, and if it gives a negative result, we escape the knockout section. In the extreme case, if all activities are passed normally, we cannot skip any activity; therefore M varies from 1 to N.

However, the *actual* processing effort can only be known once the case has completed; therefore, we approximate it by estimating the *expected* processing effort $\widehat{W_\sigma}$ of a permutation σ of knockout checks. For that we introduce the notion of reject probability. The reject probability P_i^r of a check is the probability that the given check will yield a negative outcome, i.e. knock out the case. In other words, it is the percentage of cases that do not pass the check successfully.

Let us suppose we have a knockout section with three independent checks. Table 1 lists possible scenarios during the execution of the section depending on the outcome of the checks, as well as the probabilities of these scenarios and the actually spent effort.

Table 1: Possible outcomes of checks during the execution of a knockout section with three activities.

Outcome of checks	**Probability of outcome**	**Actual effort spent**
{failed}	P_1^r	w_1
{passed; failed}	$(1-P_1^r)P_2^r$	w_1+w_2
{passed; passed; failed}	$(1-P_1^r)(1-P_2^r)P_3^r$	$w_1+w_2+w_3$
{passed; passed; passed}	$(1-P_1^r)(1-P_2^r)(1-P_3^r)$	$w_1+w_2+w_3$

Depending on the outcome of the last check, we are either leaving the knockout section proceeding with the case or terminating the case. In either situation, the processing effort would be the same. Therefore, we can join the last two scenarios and formulate the *expected* effort to execute a knockout section of three checks as:

$$\widehat{W_\sigma} = w_1P_1^r + (w_1+w_2)(1-P_1^r)P_2^r + (w_1+w_2+w_3)(1-P_1^r)(1-P_2^r) \quad (2)$$

Generalizing, the *expected* processing effort of a knockout section with N activities can be computed as follows:

$$\widehat{W_\sigma} = \sum_{i=1}^{N-1}\left(\sum_{j=1}^{i} w_j \cdot P_i^r \prod_{k=1}^{i-1}(1-P_k^r)\right) + \sum_{j=1}^{N} w_j \cdot \prod_{k=1}^{N-1}(1-P_k^r). \quad (3)$$

To estimate the expected processing effort we propose constructing predictive models for reject probabilities P_i^r and processing times T_i (see Section 3.4).

Having found the expected processing effort for all possible permutations σ of knockout activities, in our approach we select the one with the lowest expected effort. To validate the results in terms of minimizing overprocessing, we need to compare the *actual* processing effort W_σ taken after following the selected ordering σ with W_{min}.

3.3 Feature-encoding of Execution Traces

To find the reject probabilities and processing times of knockout checks we need to build the corresponding predictive models. Business process execution traces are naturally modeled as complex symbolic sequences, i.e. sequences of events each carrying data payload consisting of event attributes. However, to be suitable for conventional machine learning models, traces of completed process cases first need to be encoded in the form of feature vectors. As was found by Leontjeva et al. [12] who conducted an extensive empirical evaluation of various sequence encoding techniques and further investigated by Verenich et al. [13], an index-based encoding achieves higher accuracy and reliability when making early predictions of process outcome. Index-based encoding turns sequence into feature vectors that include both static information, coming from the case attributes and dynamic information, contained in the event payload. It is lossless encoding in the sense that all the data from the original log may be retained.

3.4 Prediction of Reject Probability and Processing Time

To make online predictions on a running case, we apply pre-built (offline) models using prefixes of historical cases before entering the knockout section. For example, if a knockout section typically starts after the n-th event, as model features we can use case attributes and event attributes of up to $(n-1)$-th event. For predicting reject probabilities of knockout activities we train classification models, while for predicting processing times we need regression models.

In order to assess the predictive power for the trained classifiers, we use the area under receiver operator characteristic curve (AUC) measure [15]. AUC represents the probability that the binary classifier will score a randomly drawn positive sample higher than a randomly drawn negative sample. A value of AUC equal to 1 indicates a perfect ranking, where any positive sample is ranked higher than any negative sample. A value of AUC equal to 0.5 indicates the worst possible classifier that is not better than random guessing. Finally, a value of AUC equal to 0 indicates a reserved perfect classifier, where all positive samples get the lowest ranks.

As a baseline, instead of predicting the reject probabilities, we use *constant* values for them computed from the percentage of cases that do not pass the particular knockout activity in the log. Similarly, for processing times of activities, we take the average processing time for each activity across all completed cases. This roughly corresponds to the approach presented in [3]. Another, even simpler baseline, assumes executing knockout activities in a random order for each case, regardless of their reject probabilities and processing times.

4 Evaluation

We implemented the proposed overprocessing prediction approach as a set of scripts for the statistical software R, and applied them to evaluate our approach using two publicly available real-life logs. In this section, we first discuss the characteristics of the two datasets. Next, we evaluate the predictive models. Finally we calculate the number of checks using the results of the prediction and compare them with the two baselines described in Section 3.4.

A package containing the R scripts, the datasets and the results of this evaluation can be downloaded from `http://apromore.org/platform/tools`.

4.1 Datasets and Features

We used two datasets derived from real-life event logs. The first log records executions of the loan origination process of *Bondora* [16], an Estonian peer-to-peer lending marketplace; the second one originates from an environmental permit process carried out by a Dutch municipality, available as part of the *CoSeLoG* project [17]. Table 2 reports the size of these two logs in terms of number of completed cases, and the rejection rate of each check. Each log has three checks, the details of which are provided next.

Bondora dataset The Bondora dataset provides a snapshot of all loan data in the Bondora marketplace that is not covered by data protection laws. These

Table 2: Summary of datasets.

Dataset	**Completed cases**	**Knockout checks**	
		Name	**Rejection rate**
Bondora	40,062	*IdCancellation*	0.080
		CreditDecision	0.029
		PostFundingCancellation	0.045
Environmental permit	1,230	*T02*	0.005
		T06	0.013
		T10	0.646

data refers to two processes: the loan origination process and the loan repayment process. Only the first process features a knockout section, hence we filtered out the data related to the second process. When a customer applies for a loan, they fill in a loan application form providing information such as their personal data, income and liabilities, with supporting documents. The loan origination process starts upon the receipt of the application and involves (among other activities) three checks: the identity check (associated with event *IdCancellation* in the log); the credit worthiness assessment (associated to event *CreditDecision*); and the documentation verification (associated to event *PostFundingCancellation*). A negative outcome of any of these checks leads to rejection of a loan application.

Bondora's clerks perform these checks in various orders based on their experience and intuition of how to minimize work, but none of the checks requires data produced by the others, so they can be reordered. Over time, the checks have been performed in different orders. For example, during a period when listing loans into the marketplace was a priority due to high investor demand, loans were listed before all document verifications had been concluded, which explains why the third check is called *PostFundingCancellation*, even though in many cases this check is performed in parallel with the other checks.

In this log, the knockout section starts immediately after the case is lodged. Thus, the only features we can use to build our predictive models are the case attributes, i.e. the information provided by the borrower at the time of lodging the application. These features are listed in Table 3, grouped into categories. A more detailed description of each attribute is available from the Bondora Web site [16]. It should also be noted that in this log there is no information about the start time and the end time of each activity. Thus, we can only use it to estimate the reject probabilities, not the processing times.

Environmental permit dataset The second dataset records the execution of the receiving phase of an environmental permit application process in a Dutch municipality [17]. The process discovered from the log has a knockout section (see Figure 2) consisting of three activities: *T02*, to check confirmation of receipt, *T06*, to determine necessity of stop advice, and *T10*, to determine necessity to stop indication. In this scenario, the checks are not completely independent. Specifically, *T10* can only be done after either *T02* or *T06* has been performed – all permutations compatible with this constraint are possible.

Another special feature of this knockout section is that in a small number of cases some checks are repeated multiple times. If the first check in a case is repeated multiple times, and then the second check is executed (and the first check is not repeated anymore after that), we simply ignore the repetition, mean-

Table 3: Features from the Bondora dataset.

Feature	**Description**
Demographical features	
Age	Age group of the borrower (0-20, 20-30, etc)
Gender	Borrower's gender (0 Male,1 Woman, 2 Undefined)
Country	Borrower's country
New	Whether or not a borrower has prior credit history
Language	Borrower's application language
Education	Borrower's education (1 Primary education, etc.)
MaritalStatus	Borrower's marital status (1 Married, etc.)
Dependents	Number of children or other dependants
Work	Employment status id (1 Unemployed, etc)
WorkTime	Employment time with the current employer
WorkExperience	Work experience in total
Area	Occupation area (2 Mining, 3 Processing, etc.)
Home	Home ownership type id (1 Owner, etc.)
Financial features	
Income	Income from principal employer
TotIncome	Total income
LiabilitiesNum	Number of liabilities before the loan
LiabilitiesMon	Total monthly liabilities
D2I	Debt to income ratio
A2I	Applied amount to income, %
L2I	Liabilities to income, %
PreviousLoanNr	Number of previous loan applications
PreviousLoanValue	Value of previous loan applications
Loan features	
Amount	Amount applied
Rate	Maximum interest rate accepted in the application
Length	The loan term
Usage	Usage of loan (101 Working capital financing, etc.)
Type	Application type (1 Timed funding, 2 Quick Funding)

ing that we treat the first check as not having been repeated by discarding all occurrences of this check except the last one. Similarly, we discarded incomplete cases as they did not allow us to assess the existence of overprocessing.

In this dataset, we could build our predictive models using both case attributes as well as the attributes of the first event, namely *Confirmation of receipt* as this event precedes the knockout section. The features derived from this log are listed in Table 4.

Table 4: Features from the Environmental dataset.

Origin	**Feature**	**Description**
case attributes	Channel	By which means the case has been lodged
	Department	Department responsible for the case
	CaseGroup	Group of the case responsible resource
	CaseResource	Resource responsible for the case
event attributes	Group	Group of the event responsible resource
	Resource	Resource responsible for the event

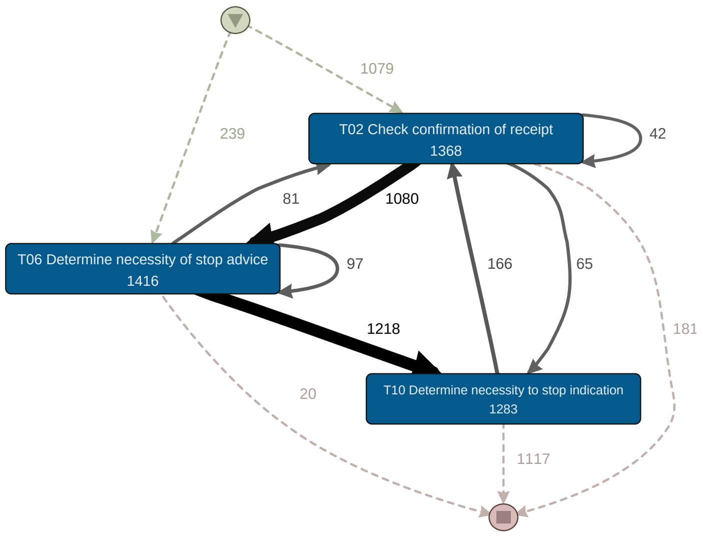

1079
239
T02 Check confirmation of receipt
1368
42
81
1080
T06 Determine necessity of stop advice
1416
97
166
65
181
1218
T10 Determine necessity to stop indication
1283
20
1117

Fig. 2: Process map extracted from the environment permit log.

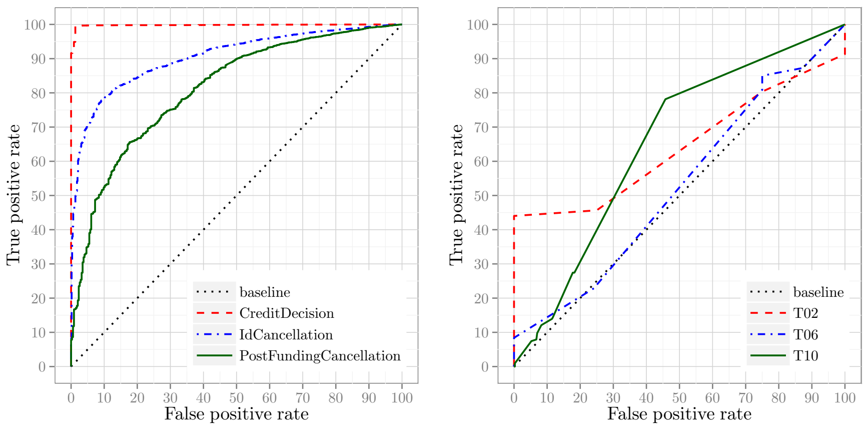
This log contains event completion timestamps but not event start timestamps. So also for this second log we do not have enough information to predict the processing time of each check, and we can only work with reject probabilities.

4.2 Assessing the Predictive Power of the Models

We split each dataset into a training set (80% of cases) to train the models, and a test set (20%) to evaluate the predictive power of the models built. As a learning algorithm we applied support vector machine (SVM) classification, trained using the `e1071` package in R. This choice allows us to build a probability model which fits a logistic distribution using maximum likelihood to the decision values of all binary classifiers, and computes the a-posteriori class probabilities for the multi-class problem using quadratic optimization [18]. Therefore, it can output not only the class label, but the probability of each class. The probability of a zero class essentially gives us an estimation of the reject probability.

It is important to notice that in both datasets the majority of cases pass all the checks successfully, thus the datasets are highly imbalanced with respect to the class labels. A naive algorithm that simply predicts all test examples as positive will have very low error, since the negative examples are so infrequent. One solution to this problem is to use a Poisson regression, which requires forming buckets of observations based on the independent attributes and modeling the aggregate response in these buckets as a Poisson random variable [19]. However, this requires discretization of all continuous independent attributes, which is not desirable in our case. A simpler and more robust solution would be to undersample positive cases. Weiss et al. [20] showed that for binary classification the optimal class proportion in the training set varies by domain and ultimate objective, but generally to produce probability estimates, a 50:50 distribution is a good option. Thus, we leave roughly as many positive examples as there are negative ones and discard the rest.

To ensure the consistency of the results we apply ten-fold cross-validation. Figure 3 shows the average ROC curves, across all ten runs. the AUC varies from 0.812 (*PostFundingCancellation*) to 0.998 (*CreditDecision*) for the Bondora dataset, and from 0.527 (*T06*) to 0.645 (*T10*) for the Environmental dataset. The lower values in the latter dataset are due to the limited number of features that can be extracted (see Table 4), as well as by the fact that the dataset has much less completed cases for training (Table 2), which is further exacerbated by having to remove many positive samples after undersampling.



(a) (b)

Fig. 3: ROC curves of predictive models for checks in *Bondora* (a) and *Environmental* (b) datasets.

4.3 Evaluation of Overprocessing Reduction

As stated in Section 3.2, the *actual* processing effort can be calculated by Formula 1. However, since the necessary timestamps are absent from our datasets, it is impossible to find the processing times T_i of the activities. Nor do we have data about the resource costs R_i. Therefore, we assume $T_i R_i = const = 1$ for all activities. Then the actual processing effort simply equals the number of performed activities in the knockout section. For both datasets, $W_\sigma = \{1, 2, 3\}$. It can be shown that in this case the optimal permutation σ that minimizes the expected processing is equivalent to ordering the knockout activities by decreasing reject probabilities.

In Table 5 we report the average number of checks and percentage of overprocessing of our approach over the ten runs, against the two baselines (constant probabilities for each check and random ordering – see Section 3.4). We found that the actual number of performed checks in case of following our suggested ordering is less than the number of checks performed in either baseline. Specifically, for the Bondora dataset we are doing only 1.22% more checks than minimally needed, which represents a 2.62 percentage points (pp) improvement over the baseline with constant probabilities and 4.51 pp improvement over the baseline with random ordering. However, for the environmental permit dataset the advantage of our approach over the constant probabilities baseline is very marginal. This can be explained by the skewed distribution of the knockout frequencies for the three checks in this dataset (the lowest knockout frequency being 0.5% and the highest being 64.6%). Thus, it is clear that the check with the lowest knockout frequency has to be executed at the end.

Table 5: Average number of performed checks and overprocessing for test cases.

	Average # of checks		**Average overprocessing, %**	
	Bondora	Environmental	Bondora	Environmental
Optimal	21,563	416	0	0
Our approach	21,828	576	1.22	38.49
Constant P_i^r's	22,393	577	3.85	38.89
Random	22,800	657	5.74	58.16

In addition, in Table 6 we report the number of cases with one, two or three knockout checks performed. As shown before, for a dataset with three checks the optimal number of checks is either one (if at least one check yields a negative outcome) or three (if all checks are passed). Therefore, in the cases with two checks, the second one should have been done first. In the Bondora dataset, such suboptimal choices are minimized; for the environmental dataset, again, our approach is just as good as the one that relies on constant probabilities.

Table 6: Distribution of number of checks across the test cases.

Ordering by	**Bondora**			**Environmental**		
	1	2	3	1	2	3
Optimal	1237	0	6775	163	0	83
Our approach	974	261	6777	2	158	86
Constant P_i^r's	642	359	7011	3	155	88
Random	413	410	7189	1	78	167

4.4 Execution times

Our approach involves some runtime overhead to find the optimal permutation as compared to the baseline scenario in which checks are performed in a predefined order. For real-time prediction it is crucial to be able to output the results faster than the mean arrival rate of cases. Thus, we also measured the average runtime overhead of our approach. All experiments were conducted using `R` version 3.2.2 on a laptop with a 2.4 GHz Intel Core i5 quad core processor and 8 Gb of RAM running the x86_64-pc-linux-gnu platform. The runtime overhead generally depends on the length of the process cases and the number of possible permutations of the checks. For the Bondora dataset, it took around 70 seconds to construct the SVM classifiers (offline) for all the checks, using default training parameters. In contrast, for the Environmental dataset with much shorter feature vectors it took less than a second to train the classifier (see Table 7). At runtime, it takes less than 2 milliseconds on average to find the optimal permutation of knockout activities for an ongoing case for both datasets (including preprocessing of the data and application of the classifier). This shows that our approach performs within reasonable bounds for online applications.

Table 7: Execution times of various components of our approach in milliseconds.

Component		Bondora		Environmental	
		mean	**st dev**	**mean**	**st dev**
Offline, overall	Learn classifier	75,000	9,000	150	20
Online, per case	Preprocess data	0.45	0.03	0.67	0.03
	Apply classifier	1.37	0.15	0.12	0.02
	Find optimal permutation	0.12	0	0.02	0

4.5 Threats to Validity

Threats to external validity are the limited number and type of logs we used for the evaluation and the use of a single classifier. While we chose only two datasets from two distinct domains (financial and government), these two datasets represent real-life logs well. They exhibit substantial differences in the number of events, event classes and total number of traces, with one log being relatively large (over 40,000 cases) and the other relatively small (around 1,200 cases).

Both the datasets used in this evaluation did not have the required start and end event timestamps to estimate the processing times of the knockout checks. Thus, we assigned a constant time to all checks. The inability to estimate processing time does not invalidate our approach. In fact, our approach would tend to further reduce the amount of overprocessing if processing times could be correctly estimated.

We reported the results with a single classified (SVM). However, we tested the approach with decision trees and random forests, and the reported results are qualitatively the same for all tested classifiers, i.e. they all improved over the baselines. We decided to only retain SVM in the paper because this classifier yielded the highest classification accuracy among all classifiers we tested. However, our approach is independent of the classifier used. Thus, using a different classifier does not in principle invalidate the results. That said, we acknowledge that the goodness of the prediction, as in any classification problem, depends on the particular classifier employed. Therefore, it is important to test multiple classifiers for a given dataset, and to apply hyperparameter tuning, in order to choose the most adequate classifier with the best configuration.

5 Conclusion and Future Work

We have presented an approach to reduce overprocessing by ordering knockout checks at runtime based on their reject probabilities and processing times determined via predictive models. Experimental results show that the proposed runtime ordering approach outperforms a design-time ordering approach when the reject probabilities of the knockout checks are close to each other. In the dataset where one check had a considerably higher rejection rate than the other, the design-time and the runtime ordering approach yielded similar results.

The proposed approach is not without limitations. One limitation of scope is that the approach is applicable when the checks are independent (i.e. can be reordered) and every check is performed once within one execution of the knockout section. In particular, the approach is not applicable when some of the knockout checks can be repeated in case of a negative outcome. This is the case for example in a university admission process, where an eligibility check may initially lead to a rejection, but the applicant can ask the application to be re-considered (and thus the check to be repeated) after providing clarifications or additional information. In other words, the current approach is applicable when a negative outcome ("reject") is definite and cannot be revoked. Similarly, we assume that a check leading to a positive outcome is definite and cannot be reconsidered. Designing heuristics for cases where the outcomes of checks are revocable is a direction for future work.

Another limitation is that the approach is designed to minimize overprocessing only, without considering other performance dimensions such as cycle time (i.e. mean duration of cases). If we add cycle time into the equation, it becomes desirable to parallelize the checks rather than sequentializing them. In other words, rather than performing the checks in a knockout section in strict sequence, some or all of checks could be started in parallel, such that whenever the first check fails, the other parallel checks are cancelled. On the one hand this parallelization leads to higher overprocessing effort, since effort is spent in partially completed checks that are later cancelled. On the other hand, it reduces overall cycle time, particularly when some of the checks involve idle time during their execution. For example, in a university admission process when some documents are found to be missing or defective, the checks involving these documents need to be put on hold until the missing or correct documents arrive. If the goal is to minimize both overprocessing and cycle time, this waiting time can be effectively used to perform other checks.

The proposed approach relies on the accuracy of the reject probability estimates provided by the classification model. It is known however that the likelihood probabilities produced by classification methods (including random forests) are not always reliable. Methods for estimating the reliability of such likelihood probabilities have been proposed in the machine learning literature [21]. A possible enhancement of the proposed approach would be to integrate heuristics that take into account such reliability estimates.

Another avenue for future work is to apply predictive methods to reduce other types of waste, such as *defect waste* induced when a defective execution of an activity subsequently leads to part of the process having to be repeated in order to correct the defect (i.e. rework). The idea is that if a defective activity execution can be detected earlier, the effects of this defect can be minimized and corrected more efficiently. Predictive process monitoring can thus help us to detect defects earlier and to trigger corrective actions as soon as possible.

Acknowledgments. NICTA is funded by the Australian Government via the Department of Communications. This research is funded by the Australian Research Council Discovery Project DP150103356 and the Estonian Research Council.

References

1. Wang, J.s.: Lean Manufacturing: Business Bottom-Line Based. CRC Press (2010)

2. Bauch, C.: Lean Product Development: making waste transparent. Master's thesis, Massachusetts Institute of Technology and Technical University of Munich (2004)
3. van der Aalst, W.M.P.: Re-engineering knock-out processes. Decision Support Systems **30**(4) (mar 2001) 451–468
4. Jansen-Vullers, M.H., Netjes, M., Reijers, H.A.: Business process redesign for effective e-commerce. In: Proceedings of the 6th international conference on Electronic commerce, ACM (2004) 382–391
5. Reijers, H.A., Mansar, S.L.: Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics. Omega **33**(4) (2005) 283–306
6. Lohrmann, M., Reichert, M.: Effective application of process improvement patterns to business processes. Software & Systems Modeling (2014)
7. Pourshahid, A., Mussbacher, G., Amyot, D., Weiss, M.: An aspect-oriented framework for Business Process Improvement. In: E-technologies: innovation in an open world. Springer (2009) 290–305
8. Niedermann, F., Radeschütz, S., Mitschang, B.: Business process optimization using formalized optimization patterns. In: Business Information Systems, Springer (2011) 123–135
9. Maggi, F.M., Di Francescomarino, C., Dumas, M., Ghidini, C.: Predictive monitoring of business processes. In: Advanced Information Systems Engineering, Springer (2014) 457–472
10. der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. Information Systems **36**(2) (2011) 450–475
11. Conforti, R., de Leoni, M., La Rosa, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: A recommendation system for predicting risks across multiple business process instances. Decision Support Systems **69** (2015) 1–19
12. Leontjeva, A., Conforti, R., Di Francescomarino, C., Dumas, M., Maggi, F.M.: Complex Symbolic Sequence Encodings for Predictive Monitoring of Business Processes. In: Business Process Management. Springer (2015) 297–313
13. Verenich, I., Dumas, M., Rosa, M.L., Maggi, F.M., Francescomarino, C.D.: Complex Symbolic Sequence Clustering and Multiple Classifiers for Predictive Process Monitoring. In: 11th International Workshop on Business Process Intelligence 2015. 1–12
14. Zeng, S., Melville, P., Lang, C.a., Boier-Martin, I., Murphy, C.: Using predictive analysis to improve invoice-to-cash collection. Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08 (2008) 1043
15. Bradley, A.P.: The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recognition **30**(7) (1997) 1145–1159
16. Bondora: Loan Dataset `https://www.bondora.ee/en/invest/statistics/data_export`. Accessed: 2015-10-23.
17. Buijs, J.: 3TU.DC Dataset: Receipt phase of an environmental permit application process (WABO) `https://data.3tu.nl/repository/uuid:a07386a5-7be3-4367-9535-70bc9e77dbe6`. Accessed: 2015-10-30.
18. Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., Leisch, F.: e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien. (2015)
19. Hill, S., Provost, F., Volinsky, C.: Network-Based Marketing: Identifying Likely Adopters via Consumer Networks. Statistial Science **21**(2) (2006) 256–276
20. Weiss, G.M., Provost, F.: Learning when training data are costly: The effect of class distribution on tree induction. Journal of Artificial Intelligence Research **19** (2003) 315–354
21. Kull, M., Flach, P.A.: Reliability Maps: A Tool to Enhance Probability Estimates and Improve Classification Accuracy. In: Machine Learning and Knowledge Discovery in Databases. Springer (2014) 18–33