AB-BPM: Performance-Driven Instance Routing for Business Process Improvement

Suhrid Satyal[1,2(✉)], Ingo Weber[1,2], Hye-young Paik[2], Claudio Di Ciccio[3], and Jan Mendling[3]

[1] Data61, CSIRO, Sydney, Australia
{suhrid.satyal,ingo.weber}@data61.csiro.au
[2] University of New South Wales, Sydney, Australia
hpaik@cse.unsw.edu.au
[3] Vienna University of Economics and Business, Vienna, Austria
{claudio.di.ciccio,jan.mendling}@wu.ac.at

Abstract. A fundamental assumption of Business Process Management (BPM) is that redesign delivers new and improved versions of business processes. This assumption, however, does not necessarily hold, and required compensatory action may be delayed until a new round in the BPM life-cycle completes. Current approaches to process redesign face this problem in one way or another, which makes rapid process improvement a central research problem of BPM today. In this paper, we address this problem by integrating concepts from process execution with ideas from DevOps. More specifically, we develop a technique called AB-BPM that offers AB testing for process versions with immediate feedback at runtime. We implemented this technique in such a way that two versions (A and B) are operational in parallel and any new process instance is routed to one of them. The routing decision is made at runtime on the basis of the achieved results for the registered performance metrics of each version. AB-BPM provides for ultimate convergence towards the best performing version, no matter if it is the old or the new version. We demonstrate the efficacy of our technique by conducting an extensive evaluation based on both synthetic and real-life data.

Keywords: Business Process Management · DevOps · AB testing · Process performance indicators

1 Introduction

Various lifecycle approaches to Business Process Management (BPM) have a common assumption that a process is incrementally improved in the redesign phase [9, Chap. 1]. While this assumption is hardly questioned in BPM research, there is evidence from the field of AB testing that improvement concepts often do *not* lead to actual improvements. For instance, work on business improvement ideas found that 75% did not lead to improvement: half of them had no impact while approximately a quarter turned out to be even harmful [12]. The results

J. Carmona et al. (Eds.): BPM 2017, LNCS 10445, pp. 113–129, 2017.
DOI: 10.1007/978-3-319-65000-5_7

are comparable to a study of the Microsoft website, in which only one third of the ideas observed had a positive impact, while the remaining had no or negative impact [15]. The same study also observed that customer preferences were difficult to anticipate before deployment, and that customer research did not predict customer behaviour accurately.

If incremental process improvement can only be achieved in a fraction of the cases, there is a need to rapidly validate the assumed benefits. Unfortunately, there are currently two major challenges for such an immediate validation. The first one is *methodological*. Classical BPM lifecycle approaches build on a labour-intensive analysis of the current process, which leads to the deployment of a redesigned version. This new version is monitored in operation, and if it does not meet performance objectives, it is made subject to analysis again. All this takes time. The second challenge is *architectural*. Contemporary Business Process Management Systems (BPMSs) enable quick deployment of process improvements, but they do not offer support for validating improvement assumptions. A performance comparison between the old and the new version may be biased since contextual factors might have changed at the same time. How a rapid validation of improvement assumptions can be integrated in the BPM lifecycle and in BPMSs is an open research question.

In this paper, we address this question by integrating business process execution concepts with ideas from DevOps. More specifically, we develop a technique called AB-BPM that offers AB testing for redesigned processes with immediate feedback at runtime. AB testing in DevOps compares two versions of a deployed product (e.g., a Web page) by observing users' responses to versions A/B, and determines which one performs better [8]. We implemented this technique in such a way that two versions (A and B) of a process are operational in parallel and any new process instance is routed to one of them. Through a series of experiments and observations, we have developed an instance routing algorithm, *LTAvgR*, which is adapted to the context of executing business processes. The routing decision is guided by the observed results for registered performance metrics of each version at runtime. The technique has been evaluated extensively on both synthetic and real-life data. The results showed that AB-BPM provides for ultimate convergence towards the best performing version.

The remainder of this paper starts with a discussion of the background and related work in Sect. 2. Section 3 describes the framework and algorithms for performing AB tests. In Sect. 4, we evaluate our approach on two use cases. In Sect. 5, we discuss the results, limitations, and validity of our approach, and finally draw conclusions in Sect. 6.

2 Background and Related Work

Business Process Management Systems (BPMSs) allow for a rapid deployment of process improvements into operation. However, there is currently no support to test the often implicit assumption that a modification of a process actually represents an improvement. One anecdote of a leading European bank (EB) illustrates this problem. The EB improved their loan approval process by cutting its turnaround time down from one week to a few hours. What happened

though was a steep decline in customer satisfaction: customers with a negative notice would complain that their application might have been declined unjustifiably; customers with a positive notice would inquire whether their application had been checked with due diligence. This anecdote emphasizes the need to carefully test improvement hypotheses in practice since customers and process participants might not act in a way that can be predicted deterministically.

Given the current architecture of BPMSs, it is not possible to conduct a fair comparison between the old and the new version of a process since they are not operational at the same point of time. That means, doing a post-hoc analysis of data generated from the old process being operational in time interval $[t(n-1), t(n)]$ and the new process running from $[t(n), t(n+1)]$ is biased towards the respective conditions of each time interval.

The need for continuous improvement is rarely disputed, but it should be complemented with the motto "test fairly" and "fail fast". This motto entails reducing the time between inception and deployment of new versions, and managing business risks brought forth by deployment of the new process versions.

From the above analysis, we derive the following three requirements:

R1 Rapidly validate the improvement assumption: a proposed improvement should be tested within a short time frame after its introduction.
R2 Ensure a fair comparison: the environment in which a comparison is conducted should minimize bias, and avoid the time bias discussed above.
R3 Enable rapid adjustments on process model level: the benefits of a solution should be suited to process models and their specific characteristics.

Concepts from DevOps [3], which aim to bring software development (Dev) and operations (Ops) closer together, may help to address this problem. One DevOps practice is live testing, where new versions of the software are tested in production with actual users of the system. The most popular form of live testing is *AB testing*, where two versions (A and B) are deployed side by side and both receive a share of the production workload while being monitored closely. The monitoring data is then used to draw conclusions about the effectiveness of one version over the other, for instance in the form of increased revenue from higher click-through rates.

So far, AB testing has been used for micro changes of websites, like changing the color of a button [8,15]. The effectiveness of this technique is surveyed by Kohavi et al. for the user interfaces of web applications [15,16]. In this paper, we adopt the idea of AB testing on the process level in order to address R1–R3. Our technique is called AB-BPM. In the following, we discuss in how far previous BPM research is related to R1–R3. We distinguish methodological and technical approaches to process improvement.

Methodological approaches include business process re-engineering and the BPM lifecycle. Business Process Re-Engineering (BPR) offers a methodology for selecting and redesigning a process to improve efficiency, often exploiting IT to support the changes [11]. BPR promotes radical changes to the processes. An explicit perspective for testing re-engineered processes is missing.

Kettinger et al. summarize methods for process improvement project [14]. Lifecycle models like the one described by Dumas et al. [9] propose a more incremental improvement of processes with periodic controlling and revisions. Our research complements this stream of research by providing techniques to experiment with process improvement hypotheses and perform statistical evaluation on them. We assume that a redesigned process is made available, so that such experimentation and analysis can be done. We envision that our approach can be used in conjunction with works that automatically generate process versions just as [4]. Other techniques include root-cause analysis [9,19], e.g. by the help of cause-effect diagrams [9,10], and the consideration of best-practises [2,20]. However, evaluation of effectiveness requires the involvement of process analysts.

Technical approaches focus on monitoring processes at runtime with a focus on specific performance metrics. Concepts based on Statistical Process Control (SPC) [13], Complex Event Processing (CEP) [24,25], and predictive analytics [6] have been proposed and adapted for monitoring business processes. However, these monitoring techniques have not been used to carry out controlled experiments. In our work, the monitoring is performed by the instance router by observing a Process Performance Indicator (PPI) like satisfaction ratings obtained from end users. Based on the chosen PPI, the instance router dynamically adjusts the request distribution rates (Table 1).

Table 1. Mapping existing works to the requirements

Approach	R1	R2	R3
Process re-engineering (Hammer/Champy) [11]	−	−	−
Process improvement [14]	−	−	−
Process lifecycle [9]	+/−	−	−
Statistical Process Control (SPC) [13]	+	−	−
Complex Event Processing (CEP) [24,25]	−	−	+
AB testing, see e.g., [3,15]	+	+	−
AB-BPM (this work)	+	+	+

Our research addresses the gap of an explicit testing of improvement hypotheses in BPM-related research and the lack of an explicit consideration of business processes in the works on AB testing. In the following, we devise our AB-BPM approach so that it meets requirements R1–R3.

3 Approach and Architecture

In this section, we present our approach, starting with mapping the instance routing problem to algorithms from the literature. Based on a small experiment, we choose one algorithm and adapt it to the context of business processes. Then we present our high-level framework, architecture, and implementation.

3.1 Instance Routing – A Multi-armed Bandit Problem

In order to integrate concepts of process execution with AB testing, we have to discuss how new instances are assigned to a specific version of the process.

Therefore, we need an instance router that distributes requests to versions in such a way that any relevant Process Performance Indicator (PPI) is maximized. The instance router also needs to deal effectively with the issue that processes can be long-running, and that PPI measurements can be delayed.

The PPI maximization can be mapped to the so-called *multi-armed bandit problem* [1,5]. The multi-armed bandit problem models a hypothetical experiment where, given some slot machines with different payoff probability distributions, a gambler has to decide on which machines to play. The objective of the gambler is to maximize the total payoff during a sequence of plays. Since the gamblers are unaware of the payoff distribution, they can approach the plays with two strategies: *exploring* the payoffs by pulling different arms on the machines or *exploiting* the current knowledge by pulling arms that are known to give good payoffs. The exploration strategy builds knowledge about the payoffs, and the exploitation strategy accumulates the payoffs. Multi-armed bandit algorithms aim to find a balance between these strategies. If the performance is affected by some context, this can be seen as the so-called *contextual multi-armed bandit problem*, where the gambler sees context (typically represented as a multi-dimensional feature vector) associated with the current iteration before making the choice.

We model the routing algorithm as a multi-armed bandit problem by representing the process versions as the "arms", and the PPI as "payoffs/rewards". The objective of the instance router is to find a good tradeoff between exploration and exploitation, possibly based on the context. To learn the performance of a version in exploration, it sends some of the process instantiation requests to either version. Based on the instance router's experience, it can exploit its knowledge to send more or even all request to the better-performing version. The reward for routing algorithm can be designed to use a PPI like user satisfaction.

3.2 Instance Routing Algorithms and Selection

The multi-armed bandit problem has been explored in related literature. LinUCB [17] is a contextual multi-armed bandit algorithm that has been employed to serve news articles to users with the objective to maximize the total number of clicks. Tompson sampling [23] is one of the simplest approaches to address multi-armed bandits. It is based on a Bayesian approach where arms are chosen according to their probability of producing optimal rewards [7,23]. Thompson sampling can be used to solve the contextual multi-armed bandit problem with linear rewards [1]. In this paper, we chose these three algorithms as candidates for process instance routing and investigate their effectiveness. We have selected these algorithms based on their demonstrated benefits and simplicity. Other algorithms, such as ϵ-greedy, ϵ-first, UCB, and EXP4 also address multi-armed bandit problems [5].

Since the goal for the routing algorithms is to maximize the cumulative value of the PPI, as preparatory work, we have experimented with different routing algorithms with different configurations to find the best performing algorithm. We have compared variations of Thompson sampling techniques [1,7,23], LinUCB [17] and a baseline algorithm which uniformly distributes requests to process versions regardless of context and rewards. We have found that *LinUCB produced the highest cumulative satisfaction score throughout the experiments.* Therefore we use this algorithm in the following. Our architecture is flexible enough that it can be easily replaced by other algorithms.

3.3 Adapting the Routing Algorithm to Business Processes

As discussed above, we chose LinUCB as our routing algorithm. However, we observed that the algorithm can be derailed by process-specific circumstances, such as the long delays before rewards. Long delays are inherent to long-running processes, and not considered in AB testing solutions for Web applications, where delays are measured in seconds or minutes. In contrast, the real-world data which we use in the evaluation has one process instance with an overall duration of more than 3 years.

This results in the following issue. Oftentimes overly long process completion times correlate with problematic process instances, leading to negative rewards. Thus, instances with short completion times can give a positive impression of a process version early on. If the algorithm receives too many positive rewards from one version during the early stages of the AB test, the algorithm is more likely to see that version as preferable. Such an early determination can introduce a bias in the evaluation. Thus, we need to ensure that the algorithm gets enough samples from both versions.

We solve this issue by adopting the idea of a "warm-up" phase from Reinforcement Learning [21], during which we emphasize exploration over exploitation. We sample the probability of exploration by using an exponential decay function, acting as an injected perturbation that diminishes as the experiment proceeds – the sample determines whether the algorithm follows LinUCB's decision or picks a version at random. We consider the "warm-up" as the experimentation phase: after all instances started during the experimentation phase are completed, no more rewards are collected and the instance router stabilizes.

Finally, the original LinUCB algorithm makes its decision based on the summation of past rewards. We found out during the experiments that this can also deceive the algorithm. Therefore, we have modified the LinUCB algorithm to make its reward estimates on the basis of the average of past rewards rather than their sum. We term our adapted instance routing algorithm *Long-term average router (LTAvgR).*

3.4 AB-BPM Framework, Architecture, and Implementation

Figure 1 shows the architecture of our AB testing framework. We designed the architecture such that the two versions of the process model are deployed side by

side in the same execution engine. The instance router distributes the instance creation requests as per its internal logic. An alternative design would be to run two full copies of the entire application stack, one for each version, and using the instance router to distribute the requests across the two stacks. However, the multi-armed bandit algorithms can identify the superior version during the experimentation and alter the allocation of requests to different versions. When a version is clearly superior to the other, most of the requests are sent to the superior version. In such scenarios, the application stack that hosts the inferior version is underutilized. If we run both versions on the same stack, we can keep utilization of the system high, no matter which version is superior.

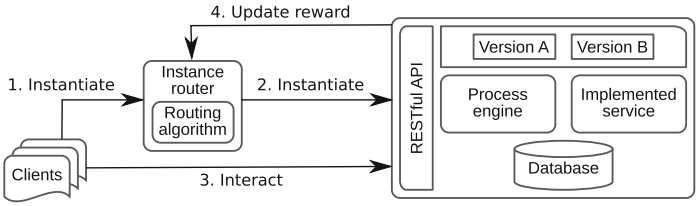


Fig. 1. Application architecture

Given this design choice, the process definitions, implementation, PPI collection, and the shared process execution engine are wrapped by a web application. Process execution data are stored in a shared database. Process instantiation, metrics, and other operations are exposed using RESTful APIs. Upon receiving a request, the instance router instantiates a particular version and receives an identifier. Identifiers of process instances for which rewards have not been observed are stored in a queue. The instance router uses a polling mechanism in parallel to retrieve PPI metrics from the server and update the rewards.

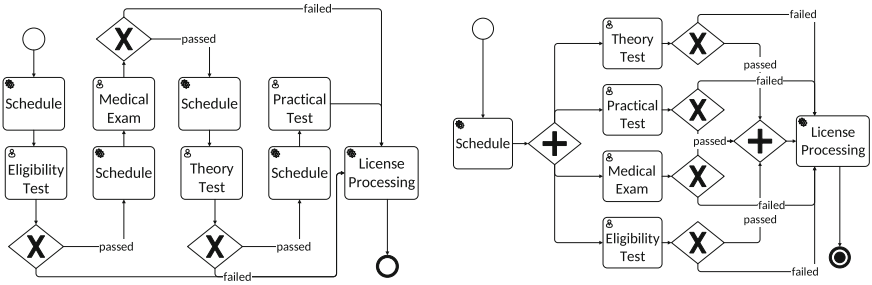
We implemented the architecture prototypically in Java and Python, in part based on the Activiti BPMS. As outlined earlier, our framework is flexible in the choice of the instance routing algorithm: we implemented and tested all five variants discussed above, i.e., LTAvgR, LinUCB, Thompson-sampling with and without linear rewards, and random uniform distribution. The experiments reported in the following section are run with the presented implementation with LTAvgR. We simulate the requests from users by replaying process logs.

4 Evaluation

In this section, we present the methodology and outcomes of our evaluation of the proposed approach. We assess the AB-BPM framework and LTAvgR algorithm first on synthetic data, where we have full control over the environment and parameters. Then we test the approach on real-world data, taken from the building permit process logs from five Dutch municipalities.

4.1 Evaluation on Synthetic Data

In this section, we demonstrate our approach using two example process versions stemming from the domain of helicopter pilot licensing. Version A of the process sequentially schedules the activities based on the cost of performing them. Based on the result (pass/fail), the process either schedules the next activity or terminates the process. In this version, we expect that successful candidates will pay more because of multiple scheduling costs. In contrast, version B of the process schedules all such activities at the beginning, thus reducing the scheduling costs. The processes are illustrated in Fig. 2. These processes have as a result the final status of the license: either *approved* or *rejected*.



(a) Version A

(b) Version B

Fig. 2. Process versions for the AB testing experiment

As PPI we simulate the user satisfaction, here calculated as a combination of cost, completion time, and result of the process execution. Costs and processing times of each task were derived from the Australian Civil Aviation Safety Authority (CASA)[1] and helicopter hiring rates from an Australian flight school.

Table 2. Cost model of the activities

Activity	Cost	Min. processing time	Max. processing time
Schedule	25	1 day	1 day
Eligibility test	190	1 day	3 days
Medical exam	75	1 day	3 days
Theory test	455	2 weeks	5 weeks
Practical test	1145	1 week	2 weeks
License processing	0 if rejected, 100 if approved	Immediate	Immediate

[1] https://www.legislation.gov.au/Details/F2016C00882, Accessed: 03-01-2017.

Table 3. User satisfaction model

Outcome	Cost	Duration	Satisfaction
Approved	$[0, 1990]$	$\leq$5 weeks	5
	$(1990, \infty]$	$\leq$5 weeks	4
	$[0, \infty)$	>5 weeks	3
Rejected	$[0, 1890]$	$\leq$5 weeks	3
	$[0, 1890]$	>5 weeks	2
	$(1890, \infty]$	$\leq$5 weeks	2
	$(1890, \infty)$	>5 weeks	1

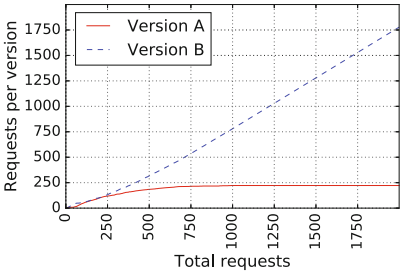


Fig. 3. Requests routing in AB tests

Table 2 shows the costs and processing times for both process versions. Table 3 shows how user satisfaction scores from 1 (lowest) to 5 (highest) are derived. The basic rationale is, the shorter and cheaper, the better. The score ranges from 1 to 3 if the outcome is negative, and from 3 to 5 if the outcome is positive.

Experiment design. We have designed the AB testing experiments such that the instance router receives requests with embedded contexts at the rate of 1 request per second. For the execution, we scale each day to 1 s. In this experiment, we introduce non-determinism in two ways: *(i)* adjusting success rates of an activity based on contextual information, and *(ii)* sampling processing times for each activity using a probability distribution function.

Results. Figure 3 shows the request distribution throughout the AB Tests. When the experimentation or "warm-up" phase ends at approximately 1000 requests, the router stops updating the reward estimates and chooses version B decisively.

A post-hoc analysis shows that the median user satisfaction was similar for both versions. The distributions of user satisfaction scores differed significantly (a Mann-Whitney test [18] resulted in $U = 54072$, p-value $<10^{-6}$ two-tailed, $n_A = 222$, $n_B = 778$). The median delay of the reward was 22.3 s. Table 4 shows the differences between the two versions. Version B produces a better user satisfaction in those cases where an application is approved. It is also faster in all cases. However, the median cost of version A is lower than that of version B when the applications are rejected.

We used an evaluation based on synthetic log data in order to investigate the convergence behaviour of the implementation. We observe that our approach leads to a rapid identification of the more rewarding process version, which receives an increasing share of traffic. This observation is instrumental with respect to the requirements R1–R3, which demand rapid validation, fair comparison and rapid adjustment on the process level.

Table 4. Analysis of versions A and B by cases

Metric	Outcome	Version A	Version B	Overall
Samples (N)	All	222	778	1000
	Approved	72	275	347
	Rejected	150	503	653
Median user satisfaction	All	3	3	3
	Approved	3	5	5
	Rejected	3	3	3
Median cost	All	795	1890	1890
	Approved	2065	1990	1990
	Rejected	795	1435	1435
Median duration	All	28.6 s	17.4 s	19.6 s
	Approved	35.5 s	21.8 s	22.4 s
	Rejected	24.5 s	8.7 s	8.9 s

4.2 Evaluation on Real-World Data

To assess the applicability of our approach over real-world data, we have analysed the data stemming from the five logs in the BPI Challenge 2015 [2], herein identified as $L^1, \ldots, L^5$. Those logs contain the execution data of the building permit issuing processes in five Dutch municipalities. The processes behind each log reportedly contain variations, which allow us to consider them as different versions of the same main process. In this experiment, we simulate the situation where one version is in use, when a new version is suggested and AB-tested in competition with the previous one. Better performance here is equated to shorter time to complete a process instance. Subsequently, the version that won the first round competes against the next version, and so on, until all versions have competed.

Based on the insights from [22], we filtered the events to retain only those activity instances that belong to a so-called "phase", namely constituting the core part of the process. Using the Inductive Miner, we discovered five process models $P^1, \ldots, P^5$ from $L^1, \ldots, L^5$, respectively. We mimicked the execution of the processes by replaying the logs on the process versions. The instance router decided to which alternative version to route each request to create a new instance. In the following, we describe how the execution times were derived, define the reward function, clarify how the competition was organized, and finally report on the achieved results.

Execution Time Simulation. For fairness, in this experiment we replay only the logs that did *not* stem from the original processes. Say, we are AB-testing

[2] BPI Challenge 2015, including logs, reports, and process models: https://www.win.tue.nl/bpi/doku.php?id=2015:challenge, Accessed 20-03-2017.

P^i vs. P^j; then we use the logs from $\mathcal{L}_{test} = \{L^1, \ldots, L^5\} \setminus \{L^i, L^j\}$ with $1 \leqslant i, j \leqslant 5$. However, we want to test how the event traces from $\mathcal{L}_{test}$ behave on P^i and P^j in terms of timing. To this end, we assign an activity duration to the execution of every replayed activity from the process version the activity was routed to. Say this is P^i, and the current activity is a; then, we randomly sample a duration value from the durations of a among all the traces of L^i sharing the same execution history (prefix) as the replayed trace.

For instance, consider the execution of activity *phase concept draft decision ready* for process P^1 after the sub-trace [*phase application received*, *phase application receptive*, *phase advice known*]. This activity has been assigned with a random sample among the registered execution times of *phase concept draft decision ready*-events following [*phase application received*, *phase application receptive*,*phase advice known*] in the 593 traces of log L^1 sharing that prefix.

To that extent, we have folded the traces of every log into a dedicated poly-tree auxiliary data structure, collapsing the traces that share the same prefix on common paths. Every node keeps the activity name and the list of registered execution times of the related events. In addition, event transitions in the log are stored in a table structure along with the list of transition times. When the traces cannot be followed in the poly-tree structure, we perform a lookup on the table structure and derive execution times. If the current transition has not been observed in L^i, we discard the trace as non-conforming and disregard it in the reward calculation.

The events in the logs signal the completion of an activity, and bear eight timestamp fields. However, most of those attributes were missing or unreliable. Therefore, we followed the approach of [22], and used solely the completion *time:timestamp* attribute for each event. We computed the duration of every activity as the difference between the timestamp of its completion and the preceding completion timestamp. We thus included in the activities' duration estimation both the execution time and the waiting time before starting.

Reward Strategy. The filtered BPIC 2015 dataset contains numerous outliers: while the median duration for processes are 39–46 days, outliers can take up to 1158 days, i.e., 3 years and 63 days. To establish a reward function that penalizes very long process completion times, we adopted the following strategy. Given the initial version P^i of a process, we collected all instance execution times reported in its log L^i and computed $K \geqslant 1$ quantiles $q_1, \ldots, q_K$. We used these quantiles to partition the space of possible execution times into a set of $K + 2$ intervals $I = \{\iota_0, \iota_1, \ldots, \iota_K, \iota_{K+1}\}$ where $\iota_0 = [0, q_0)$, $\iota_{K+1} = [q_K, +\infty)$, and $\iota_k = [q_{k-1}, q_k)$ for $1 \leqslant k \leqslant K$. Those intervals split the range of possible execution times for P^j as follows: ι_0 contains the values below the minimum registered in L^i, ι_{K+1} accounts for any duration

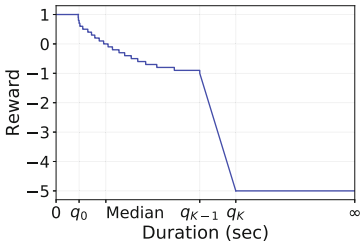


Fig. 4. Reward strategy

Algorithm 1. Strategy for the selection of the best performing version among $\{P^1, P^2, P^3, P^4, P^5\}$.

```
P_test ← {P^1, P^2, P^3, P^4, P^5}
P^i ← original process version from P_test
P_test ← P_test \ {P^i}
repeat
    P^j ← alternative process version from P_test
    P_test ← P_test \ {P^j}
    L_test ← {L^1, ..., L^5} \ {L^i, L^j}
    P^i ← best version between P^i and P^j as per the AB test over L_test
until P_test ≠ ∅
return P^i
```

beyond the maximum recorded in L^i, and the K intervals in-between are meant to classify the performance of P^j with respect to their quantile as per L^i. This strategy is illustrated as a step chart in Fig. 4 – every step in the chart represents a quantile.

The idea is to assign a reward of 1 when the duration P^j achieves is lower than any registered execution time of P^i, and decrease it by a step of $2/K$ as long as the measured performance falls into the following intervals. In order to counterbalance the disruptive effect of outliers which take extremely long, we established a penalty value ρ (with $\rho = 4$ in Fig. 4) and defined that the reward linearly decreases from -1 to $(-1-\rho)$ along ι_K. Finally, any execution time beyond the last quantile is assigned a reward of $(-1-\rho)$.

Formally, let $\kappa_I(t) : \mathbb{R}^+ \to [0, \ldots, K+1]$ be a mapping function associating a process instance execution time t to the respective interval $\iota_K \in I$ by the index k, e.g. $\kappa_I(t) = 3$ if t is in the range of ι_3. The reward function $r_I : \mathbb{R}^+ \to [-1-\rho, 1]$ is defined over the set of intervals I as follows:

$$r_I(t) = \begin{cases} 1 - \frac{\kappa_I(t)}{K} \cdot 2 & \text{if } \kappa_I(t) < K \\ -1 - \rho \cdot \frac{t - q_{K-1}}{q_K - q_{K-1}} & \text{if } \kappa_I(t) = K \\ -1 - \rho & \text{if } \kappa_I(t) > K \end{cases} \quad \text{with} \quad \kappa_I(t) = \sum_{k=0}^{K+1} k \cdot \chi_{\iota_k}(t)$$

where χ_{ι_k} is the characteristic function of interval ι_k. The underlying idea was to prefer the process demonstrating a shorter completion time to the slower ones while accounting for the outliers. For our experiment, we set $K = 20$ and $\rho = 4$.

Competition: Selecting the Best Version. To simulate the situation where an organization gradually designs new versions of a process model, we run a competition between the five provided process models. This competition is conducted as a set of pair-wise comparisons between versions, following the schema outlined in Algorithm 1. The idea is to initially consider an original version of the process, P^i, and a new version, P^j. To determine if P^j achieves an actual improvement over P^i while limiting bias as discussed above, the execution of the processes is simulated by replaying the traces in the logs from which P^i and

Table 5. Number of traces

Log	Traces
L^1	1199
L^2	830
L^3	1409
L^4	1051
L^5	1155

Table 6. Ratio of conforming traces

Version	L^1	L^2	L^3	L^4	L^5
P^1	1	0.928	0.949	0.974	0.928
P^2	0.913	1	0.928	0.982	0.938
P^3	0.901	0.812	1	0.975	0.886
P^4	0.873	0.731	0.913	1	0.829
P^5	0.897	0.929	0.944	0.979	1

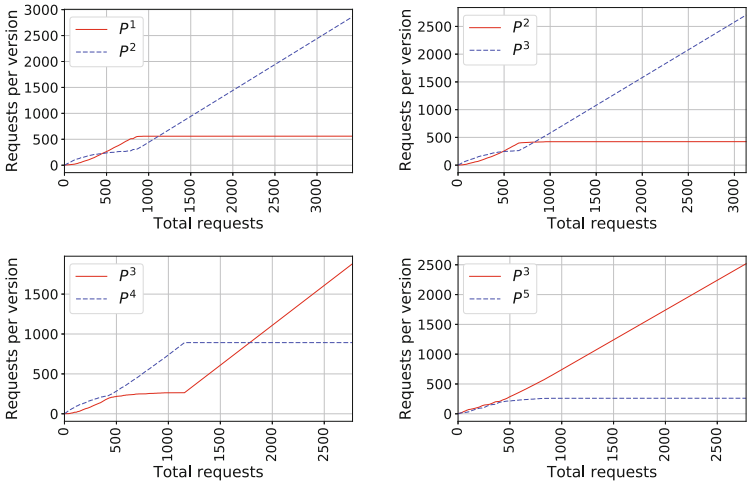


Fig. 5. Request distribution over time

P^j were *not* derived. For instance, P^1 and P^2 are evaluated on the basis of the traces in L^3, L^4, and L^5. If, at the end of a competition round, P^j demonstrated an improvement over P^i, then P^i is replaced with P^j. Otherwise, P^i is maintained. At that stage, another process version is compared to P^i. The selection procedure continues until all process versions have competed. We remark here that the traces which could not be replayed on the process picked by the instance router were discarded. The number of compliant traces still represents the vast majority, because the ratio of conforming traces of all logs over models remained around 0.9, and always above 0.7 as shown in Table 6. Also, the total number of traces per log is shown in Table 5.

Analysis. Without loss of generality, we began the selection considering P^1 as the process currently running on the production system, and progressively entering P^2, P^3, P^4, and P^5 into the competition as described above. We sped up the execution time such that one day in the trace was equated to one second in the experiments.

The sequence of tests was: (1) P^1 vs. P^2, P^2 wins. (2) P^2 vs. P^3, P^3 wins. (3) P^3 vs. P^4, P^3 wins. (4) P^3 vs. P^5, P^3 wins. We can observe that P^3 was the best-performing version. In all tests, the instance router chose the version with lower mean and median execution time.

Figure 5 shows the request distribution throughout the pair-wise tests. The experimentation phase ends roughly after 1000 requests in all cases. We can observe that occasionally the instance router decided to pick another version some time during the post-experimentation phase. In some cases, the decision made during the post-experimentation phase contradicted the decision during the experimentation phase. In these scenarios, the instance router was able to make the better decision only after all the delayed rewards were received.

In Table 7, we show the request distribution during the experimentation phase, and the performance metrics calculated using execution times of processes instantiated during this phase. Considering the median and mean times in this table confirms that the instance router using the *LTAvgR* algorithm made *the right decision in all cases.*

Table 7. Pair-wise performance comparison of versions after the AB tests

Metric	Round 1		Round 2		Round 3		Round 4	
	P^1	P^2	P^2	P^3	P^3	P^4	P^3	P^5
No. of requests	559	440	423	575	263	729	735	261
Median duration	33.8	29.8	28.8	27	21	21.85	22.9	27.9
Mean duration	55.3	52.1	51.8	35.8	29.3	49.9	36.6	38.3

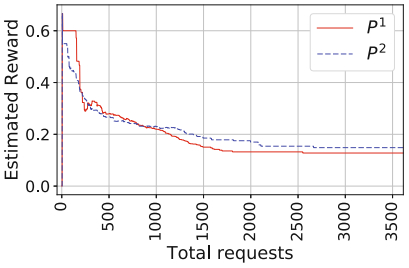


Fig. 6. Estimated rewards during the experiment P^1 vs. P^2

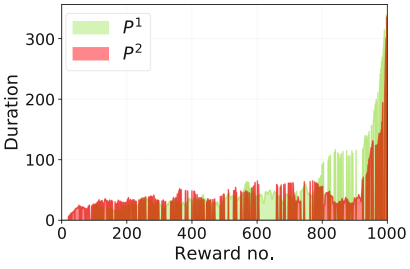


Fig. 7. PPI (duration) during the warm-up phase, smoothed.

For an in-depth view of the reward estimates (the average reward observed by *LTAvgR*) and execution times, we depict in Figs. 6 and 7 how their values changed during the experiment P^1 vs. P^2. The effect that fast completion leads to positive rewards is clearly visible in Fig. 6: shortly after the start of the experiment, the reward estimates for both versions jump to more than 0.6. After some

fluctuation, P^1 is preferred approximately from request 280 to request 811. This is also visible in Fig. 5, where we can observe that the change in maximum of the two reward estimates leads to change in the request distribution strategy.

Figure 7 shows the PPIs observed by the instance router in order. Better PPIs, which lead to better rewards, are received early. However, worse PPIs tend to accumulate near the end of the warm-up phase. At request 811, the two estimated rewards are very close to each other – see Fig. 6. At this point, P^2 collects actual rewards from longer durations than P^1 – see Fig. 7. These longer durations result in negative rewards, which cause the reward estimate of P^2 to fall below that of P^1. This development leads to the change in the decision of *LTAvgR*.

5 Discussion

The design of our routing algorithm, *LTAvgR*, was informed by practical observations of the limitations when applying existing algorithms in the process execution context. As we have demonstrated, our approach addresses the key requirements R1-R3. Our evaluation focused on the practical use of AB-BPM and *LTAvgR*; theoretical analyses of the routing algorithm were out of the scope. The in-depth analysis above showed how *business-relevant PPI observations have a direct influence* on the routing decisions taken by *LTAvgR*.

We have used a multi-armed bandit algorithm with rewards derived from a single PPI. In practice, however, multiple PPIs may need to be considered. Furthermore, optimizing routing for one PPI can negatively affect other PPIs. One key challenge in using multiple PPIs is that the reward delay for each PPI can be different. Dealing with such scenarios may require improved collection and reward update mechanisms, which we plan to explore in future work.

One limitation of our evaluation of AB-BPM so far is that they are based on isolated environments with no real user interactions. Factors like effects from the novelty of a process version were not considered. For example, in changing the user interfaces and forms, we may observe that users behave differently when exposed to a new version. As with the case study using real-world logs, we expect to find some patterns unique to business processes when these factors are accounted for. We believe that observations from real-world systems can guide us towards designing a better instance routing algorithm, and identifying best practices for performing AB tests on process versions.

6 Conclusion

Business process improvement ideas do not necessarily manifest in actual improvements. In this paper we proposed the AB-BPM approach which can rapidly validate process improvement efforts, ensure fair comparison, and make process level adjustments in production environments. Our approach uses an

instance router that dynamically selects process versions based on their historical performance in terms of the chosen PPI. To this end, we proposed the *LTAvgR* algorithm that can cater for the specifics of business process execution.

We evaluated our approach through synthetic and real-world data. We chose the most effective routing algorithm based on a set of experiments, and evaluated business process versions with synthetic data. Further, we evaluated real-world process versions by performing pair-wise AB tests on them. The evaluation results showed that our instance router dynamically adjusted request distribution to favour the better performing version.

In future work, we aim to integrate our framework with approaches to balance multiple PPIs. In addition, we plan to consider user interaction, and run industrial case studies where we apply our instance router to actual production systems.

Acknowledgements. The work of Claudio Di Ciccio has received funding from the EU H2020 programme under the MSCA-RISE agreement 645751 (RISE_BPM).

References

1. Agrawal, S., Goyal, N.: Thompson sampling for contextual bandits with linear payoffs. In: International Conference on Machine Learning, ICML (2013)
2. Alter, S.: Work system theory: overview of core concepts, extensions, and challenges for the future. J. Assoc. Inf. Syst. **14**, 72 (2013)
3. Bass, L., Weber, I., Zhu, L.: DevOps: A Software Architect's Perspective. Addison-Wesley Professional, New York (2015)
4. Burattin, A.: PLG2: multiperspective processes randomization and simulation for online and offline settings. CoRR abs/1506.08415 (2015)
5. Burtini, G., Loeppky, J., Lawrence, R.: A survey of online experiment design with the stochastic multi-armed bandit. CoRR abs/1510.00757 (2015)
6. Cabanillas, C., Di Ciccio, C., Mendling, J., Baumgrass, A.: Predictive task monitoring for business processes. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 424–432. Springer, Cham (2014). doi:10.1007/978-3-319-10172-9_31
7. Chapelle, O., Li, L.: An empirical evaluation of Thompson sampling. In: Neural Information Processing Systems (NIPS) (2011)
8. Crook, T., Frasca, B., Kohavi, R., Longbotham, R.: Seven pitfalls to avoid when running controlled experiments on the web. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1105–1114 (2009)
9. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer, Heidelberg (2013)
10. Gregory, F.: Cause, effect, efficiency and soft systems models. J. Oper. Res. Soc. **44**, 333–344 (1993)
11. Hammer, M., Champy, J.: Reengineering the Corporation: A Manifesto for Business Revolution. HarperCollins, New York (1993)
12. Holland, C.W.: Breakthrough Business Results with MVT: A Fast, Cost-Free "Secret Weapon" for Boosting Sales, Cutting Expenses, and Improving Any Business Process. Wiley, Hoboken (2005)

13. Jiang, W., Au, T., Tsui, K.L.: A statistical process control approach to business activity monitoring. IIE Trans. **39**(3), 235–249 (2007)
14. Kettinger, W.J., Teng, J.T.C., Guha, S.: Business process change: a study of methodologies, techniques, and tools. MIS Q. **21**(1), 55–98 (1997). http://dblp.uni-trier.de/rec/bib/journals/misq/KettingerTG97
15. Kohavi, R., Longbotham, R., Sommerfield, D., Henne, R.M.: Controlled experiments on the web: survey and practical guide. Data Min. Knowl. Discov. **18**(1), 140–181 (2009)
16. Kohavi, R., Crook, T., Longbotham, R., Frasca, B., Henne, R., Ferres, J.L., Melamed, T.: Online experimentation at Microsoft. In: Workshop on Data Mining Case Studies (2009)
17. Li, L., Chu, W., Langford, J., Schapire, R.E.: A contextual-bandit approach to personalized news article recommendation. In: International Conference on World Wide Web (2010)
18. Mann, H.B., Whitney, D.R.: On a test of whether one of two random variables is stochastically larger than the other. Ann. Math. Stat. **18**, 50–60 (1947)
19. Ōno, T.: Toyota Production System: Beyond Large-scale Production. Productivity Press, Portland (1988)
20. Poelmans, S., Reijers, H.A., Recker, J.: Investigating the success of operational business process management systems. Inf. Tech. Manage. **14**(4), 295–314 (2013)
21. Sutton, R.S., Barto, A.G.: Introduction to Reinforcement Learning, 1st edn. MIT Press, Cambridge (1998)
22. Teinemaa, I., Leontjeva, A., Masing, K.O.: BPIC 2015: diagnostics of building permit application process in Dutch municipalities. BPI Challenge Report 72 (2015)
23. Thompson, W.R.: On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometrika **25**(3/4), 285–294 (1933)
24. Weidlich, M., Ziekow, H., Gal, A., Mendling, J., Weske, M.: Optimizing event pattern matching using business process models. IEEE Trans. Knowl. Data Eng. **26**(11), 2759–2773 (2014)
25. Weidlich, M., Ziekow, H., Mendling, J., Günther, O., Weske, M., Desai, N.: Event-based monitoring of process execution violations. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 182–198. Springer, Heidelberg (2011). doi:10.1007/978-3-642-23059-2_16