AB Testing for Process Versions with Contextual Multi-armed Bandit Algorithms

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

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

Abstract. Business process improvement ideas can be validated through sequential experiment techniques like AB Testing. Such approaches have the inherent risk of exposing customers to an inferior process version, which is why the inferior version should be discarded as quickly as possible. In this paper, we propose a contextual multi-armed bandit algorithm that can observe the performance of process versions and dynamically adjust the routing policy so that the customers are directed to the version that can best serve them. Our algorithm learns the best routing policy in the presence of complications such as multiple process performance indicators, delays in indicator observation, incomplete or partial observations, and contextual factors. We also propose a pluggable architecture that supports such routing algorithms. We evaluate our approach with a case study. Furthermore, we demonstrate that our approach identifies the best routing policy given the process performance and that it scales horizontally.

Keywords: Multi-armed bandit, Business Process Management, AB Testing, Process Performance Indicators

1 Introduction

Business improvement ideas often do *not* lead to actual improvements [9,10]. Contemporary Business Process Management Systems (BPMSs) enable quick deployment of new process ideas, but they do not offer support for validating the improvement assumptions existent in the new version. Support for validating such assumptions during process redesign is also limited.

The AB testing approach from DevOps can be adopted in Business Processes Management to provide fair validation support. A new process version can be deployed alongside the older version on the same process engine such that these versions (A and B) are operational in parallel. User requests can be routed to either of these versions using various instance routing algorithms. Based on the performance of each version, the routing configuration can be dynamically

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-319-91563-0_2

adjusted to ultimately find the best performing version. This general idea has been introduced in AB-BPM [14]. However, the routing algorithm proposed in AB-BPM does not address scenarios where the process performance is measured through multiple Process Performance Indicators (PPIs) which may be available at different times. It also does not provide support for evaluating processes for which some of these PPIs may never be available, and processes that are affected by external factors (e.g. the weather condition).

In this paper, we address these shortcomings by revising the routing mechanism of AB-BPM. We propose a pluggable instance router architecture that allows routing algorithms to asynchronously collect and evaluate PPIs. We also propose a routing algorithm, *ProcessBandit*, that can be plugged into the instance router. ProcessBandit finds a good routing policy in the presence of delays and incompleteness in observing PPIs, and also when true performance depends on contextual external conditions. We show that our approach identifies the best routing policy given the performance, and that it scales horizontally. We also demonstrate the overall approach using a synthetic case study.

The remainder of the paper starts with a discussion on the background, key requirements, and related work in Section 2. Section 3 describes the architecture of the instance router and the details of ProcessBandit algorithm. In Section 4, we analyze the behaviour of the algorithm, and study a use case. Section 5 discusses our approach and draws conclusions.

2 Background

2.1 Problem Description and Requirements

Business process improvement efforts are often analysed by measuring four performance dimensions: time, cost, flexibility, and quality. Improvement decisions have to reflect trade-offs between these dimensions [7,13]. In many cases, shortcomings in one dimension may not be compensated by improvements in other dimensions. For example, a low user satisfaction cannot be compensated with faster performance. In addition, the relationships between these dimensions may not be intuitive. This is illustrated by an anecdote of a leading European bank. The bank improved their loan approval process by cutting turnaround time down from one week to few hours. However, this resulted in 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 shows that customer preferences are difficult to anticipate before deployment and that there is a need to carefully test improvement hypotheses in practice. Up until now, only the AB-BPM approach [14] supports the idea of using principles of AB testing to address these problems. As an early proposal, AB-BPM has a number of limitations regarding utilization of PPIs.

First of all, there may be multiple process performance indicators involved in determining a better version. For instance, both user satisfaction score and

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-319-91563-0_2

Table 1: Requirements of AB testing system and our approach

	Requirement	Approach
R1	Performance of a process execution is determined by multiple PPIs	Reward design encapsulating all PPIs
R2	Individual PPIs are available at different times	Asynchronously fetch PPIs and update rewards
R3	Most process instances do not provide all PPIs	Maintain ratio of complete and incomplete rewards
R4	Performance of a process instance is affected by *contextual factors*	Identify and integrate contextual factors in the algorithm

process execution time are acceptable PPIs for a process instance. Second, not all of the required Process Performance Indicators (PPIs) may be observable at the same time. A PPI such as the user satisfaction score is obtained at different times with delays of varying length. This means that the evaluation mechanism should support a PPI to be collected and aggregated asynchronously, i.e., at different points in time. Finally, some process instances will not produce all of the PPIs. It is likely, for example, that the number of users who do not respond to requests for providing satisfaction scores will outnumber those who do. Therefore, we should also be able to handle the missing or incomplete PPI observations.

Another aspect to consider is the effect of *contextual factor*s. The performance of a process can be influenced by factors such as resource constraints, the environment, and market fluctuation. One example of influence of weather has been observed in "teleclaims" process of an insurance company [1]. The call centers of this company receive an incoming call volume of 9,000 per week. However, during a storm season, the volume can reach up to 20,000 calls per week. In order to manage this influx, the managers manually escalate the cases to maintain quality and meet deadlines. Identifying and acting on such contextual factors is crucial to find the best process version.

From the above analysis, we derive four key requirements and propose approaches outlined in Table 1. To address these requirements, we have implemented a two-pronged solution: a pluggable architecture for instance routing, and a routing algorithm that asynchronously learns about process performance.

2.2 Related Work

AB testing is a commonly used approach for performing randomized sequential experiments. This approach is widely used to test micro changes in web applications [6,10,11]. In applying AB testing to business process versions, performing randomized experiments can inadvertently introduce risks such as loss of revenue. Risks in this context are higher than that for standard web applications, where the changes and the effects are small (e.g. the placement of buttons). Therefore, user requests should be distributed according to performance of process versions.

We introduced the idea of AB testing for business process versions in AB-BPM [14], where we modeled this routing challenge as a *contextual multi-armed*

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-319-91563-0_2

bandit problem [4,12,2]. We proposed LtAvgR, which is based on LinUCB [12,5] – a well-known contextual multi-armed bandit algorithm. LtAvgR dynamically adjusts how user requests are routed to each version by observing numerical rewards derived from process performance. LtAvgR defines an experimentation phase where observing rewards is emphasized over optimal routing, and a post-experimentation phase where the best routing policy is selected based on the observed rewards. LtAvgR updates its learning by averaging historical rewards, which enables it to support long-running processes. However, LtAvgR can only handle scenarios where all PPIs are available at the same time. In this paper, we propose *ProcessBandit*, an algorithm that addresses this limitation.

Multi-armed bandit algorithms have been adopted for various kinds for experiment designs [4]. However, work on the effect of feedback delay and impact of partial rewards is not well studied. Furthermore, the effect of sparseness of some rewards, such as those with user satisfaction scores, have not been considered for multi-armed bandits. Temporal-difference (TD) learning can be used to converge towards the best routing configuration in presence of delayed rewards [17, Ch 6]. Silver et al. [16] propose an asynchronous concurrent TD learning approach for maximizing metrics such as customer satisfaction by learning from partial customer interactions. This approach can be used in sequential scenarios like marketing campaigns where interactions can affect customer state. We propose a simpler multi-armed bandit algorithm that handles asynchronous learning with partial rewards, adapted for scenarios where only one interaction (processes instantiation) needs to be observed.

Approaches for prediction based on imbalanced data include techniques such as oversampling and undersampling [8, Ch 2]. Such techniques make assumptions about what balanced data should look like, and do not introduce any new knowledge. In our scenario, imbalance occurs when only a subset of all PPIs are observed. Since AB testing aims to remove implicit assumptions, we avoid sampling techniques. Instead, we ensure that the routing algorithm learns mostly through observations that have all PPIs.

3 Solution

Our solution consists of two parts. First, we propose a pluggable and scalable architecture that facilitates a routing algorithm to learn the best routing policy even when PPIs are missing, delayed, or incomplete, and when the performance is affected by contextual factors. Second, we propose a routing algorithm named *ProcessBandit* that learns routing policies by utilizing this architecture.

3.1 The Instance Router

The instance router is a modular system composed of an Asynchronous Task Queue, the Controller, the Routing Algorithm, the Context Module, the Tasks Module, and the Rewards Module. Figure 1 shows the architecture of the system.

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-319-91563-0_2

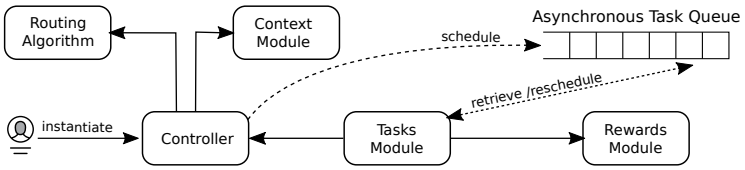

Fig. 1: The architecture of instance router

The instance router assigns an instance of the deployed processes, version A or B, to an incoming request. Upon receiving a request, the Controller invokes the Context Module to extract contextual information from the request and construct a feature vector. If required by the Routing Algorithm, the Context Module captures and stores *hypothesized* contextual factors associated with each request. These hypothesized contextual factors are set at the start of AB tests. If a contextual factor is confirmed through the analysis of the stored values, the Context Module integrates the contextual factor in the feature vector. Using this feature vector, the Controller invokes the Routing Algorithm, which instantiates a process and returns an identifier. This process instance identifier is used by the Controller to schedule an update task on the Asynchronous Task Queue.

An update task asynchronously polls the BPMS for PPIs of the instantiated process, and calculates a numerical reward using the PPIs. When only a subset of the desired PPIs are available, update tasks are re-scheduled by the Tasks Module so that that the missing PPIs can be collected and evaluated at a later point in time. In such scenarios, temporary rewards are calculated using the available PPIs. Reward calculation is delegated to the Rewards Modules. The Routing Algorithm can learn a routing policy through these numerical rewards.

3.2 ProcessBandit Algorithm

We propose *ProcessBandit*, a routing algorithm that can be plugged into the architecture. The algorithm asynchronously observes PPIs associated with a particular request, distributes requests to process versions, observes process performance, and learns the best routing policy given the process performance.

The pseudo code for sampling a process version (or "arm" in multi-armed bandit terminology) to test its performance is shown in Algorithm 1. The algorithm maintains an average of complete, incomplete, and overall rewards for each d-dimensional context in relevant matrices, indicated as b. These values are updated asynchronously according to the performance of each process instance.

The algorithm consists of experimentation (P1) and post-experimentation (P2) phases. When contextual factor detection is enabled, the experimentation phase is further divided into pre-contextual factor (P1A) and post-contextual factor (P1B) phases. The phases are configured using an exponential decay function $exp(\lambda)$, experimentation request threshold M, and pre-contextual factor reward threshold. Request count is incremented on process instantiation. Reward count is incremented when a reward calculated using all PPIs is received.

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-319-91563-0_2

Algorithm 1: ProcessBandit Instance Routing

Input: $\alpha \in \mathbb{R}_+$, $\lambda \in \mathbb{R}$, $M \in \mathbb{N}$
// α is the LinUCB's tuning parameter; λ and M are experimentation decay and length
Output: arm id

$I \leftarrow$ empty set // set of contextual factors
for $t = 1, 2, 3, \ldots, T$ **do** // t is the request count
  Observe features of all arms $a \in A_t : x_{t,a} \in \mathbb{R}^d$
  constructFeatureVector(I) // Algorithm 3
  for $a \in A_t$ **do**
    if a *is new* **then**
      $A_a \leftarrow I_d$, $b_a \leftarrow 0_d$ // identity and zero matrices of dimension $d \times d$, resp.
    $\hat{\theta}_a \leftarrow A_a^{-1} b_a$
    $p_{t,a} \leftarrow \hat{\theta}_a^\intercal \; x_{t,a} + \alpha\sqrt{x_{t,a}^\intercal \; A_a^{-1} \; x_{t,a}}$
  arm $a_{\text{linucb}} = \underset{a \epsilon A_t}{\text{argmax}}\, p_{t,a}$ with ties broken arbritarily
  if $t \leqslant M$ **then** // experimentation phase
    $pr_{\text{exp}} \leftarrow$ sample y from $Exp(\lambda)$ s.t. $x = t$
    Choose arm $a_t = a_{\text{linucb}}$ or $a_{\text{alternate}}$ with probability pr_{exp}
    Schedule update task
  else
    Choose arm $a_t = a_{\text{linucb}}$

The algorithm uses an approach similar to LinUCB [12] to select a candidate arm a_{linucb} such that the expected reward is maximized. When the algorithm is in experimentation phase, it either chooses a_{linucb} or the alternate arm based on the probability sampled from the exponential decay function. Asynchronous reward updates are scheduled for all decisions made in the experimentation phase.

Asynchronous Reward Update. We define an ideal PPI vector p_{ideal} as the vector that represents the best possible values for each PPI. We also introduce a reference vector p_{ref}, which defines values that can be used as a substitute for missing PPIs. In AB testing scenarios, historical data of one process version is available. This can be used to inform the choice of p_{ref}. Finally, we define the effective vector p_{eff} as the vector that contains all PPIs used to evaluate a reward. If not all PPIs are available at the time of observing a completed process instance, an effective vector p_{eff} is constructed using components of reference vector p_{ref} instead of the unavailable PPIs. If/when these PPIs are made available, an update is applied by removing the effect of previous p_{eff}, and then using the new effective vector. This helps us address requirements *R1* and *R2*.

Using p_{eff}, rewards can be calculated through a *point-based* or *classification based approach*, as illustrated in Fig. 2. In the point-based method, the Rewards Module constructs p_{eff}, applies weights to PPIs (if any), and then normalizes all components of p_{eff} and p_{ideal}. After the normalization, it calculates the effective reward as the euclidean distance between p_{eff} and p_{ideal}. Therefore, the objective of the algorithm is to choose versions that produce shorter distances between the effective vector and the ideal vector.

The point-based approach is intuitive and easy to implement. However, it makes the implicit assumption that a decrease in one PPI can be compensated by an increase in another PPI [3, Ch 2]. In many real-world scenarios, this may not be the case. For example, while the increase in costs may be compensated with better processing times, lower user satisfaction may not be compensated with

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-319-91563-0_2

Algorithm 2: Asynchronous Update

Input: τ, x_{t,a_t}, A_{a_t}, b_{a_t} // τ: ratio of (in)complete rewards, others as in Alg. 1

$A_{a_t} \leftarrow A_{a_t} + x_{t,a_t}\ x_{t,a_t}^t$
Construct p_{eff} and derive reward
begin with context x_{t,a_t}
  $r_{\text{incomplete}} \leftarrow$ avg. incomplete reward, $r_{\text{complete}} \leftarrow$ avg. complete reward
  $r_{\text{avg}} \leftarrow$ average overall reward
  if old reward **then** update $r_{\text{incomplete}}$, and r_{complete}
  if new reward **then**
    if reward ratio $\geqslant \tau$ **or** bootstrap period **then**
      if all PPIs seen **then** update r_{complete} , increment r_{complete} count
      else update $r_{\text{incomplete}}$, increment $r_{\text{incomplete}}$ count
    else
      if all PPIs seen **then** update r_{complete} , increment r_{complete} count
      else update $r_{\text{incomplete}}$ as moving average
  update r_{avg}
Update b_{a_t} such that r_{avg} represents x_{t,a_t}

any other metric. In addition, granular and insignificant differences in distance can accumulate and produce an effect on routing. In such scenarios, a better approach is to classify performance into categories aligned with business goals.

Therefore, in the classification-based approach domain experts design reward classes and assign weights to each class. Weights represent the relative importance of each class. p_{eff} is constructed as above and a reward is assigned as the weight of the class it falls on. The most important class C_i has the highest weight w_i. As depicted in Fig. 2, the most important class C_1 has the highest weight w_1, C_2 has a lower weight w_2, and so on. The objective of the algorithm is to choose versions that produce the highest average weight.

The algorithm specification is independent of how reward values are derived from PPIs. Without loss of generality we typically choose rewards on a negative scale (e.g. $w_1 \mapsto -1, \ldots, w_5 \mapsto -5$). The learning rate and convergence are, however, dependent on the quantity of the reward. It is possible for the algorithm to be misled by a large quantity of rewards derived from partially observed metrics. If only a small percentage of process instances provide information about all PPIs, the effect of rewards derived from these process instances can be diluted by rewards derived using incomplete PPI observations from other process instances. To ensure that such dilution does not occur, the algorithm keeps track of the *ratio of complete and incomplete rewards*, τ, for each version in each context. To accommodate τ, Algorithm 2 starts in bootstrap mode for the first few requests. During bootstrap, rewards are collected regardless of τ. The algorithm accepts a partial reward either at the bootstrap period when the number of complete rewards is below a certain threshold, or when the reward ratio is less than or equal to τ. The usage of reward ratios in this manner addresses requirement *R3*.

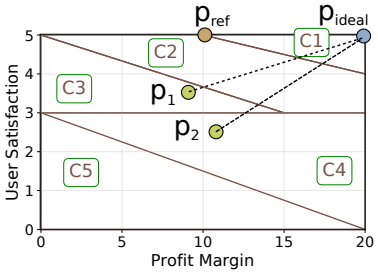

Fig. 2: Reward design approaches. Rewards can be categorical or distance based.

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-319-91563-0_2

Algorithm 3: Contextual Factor Detection

```
1  def constructFeatureVector(I)                    // I: set of contextual factors
2      if reward count < pre-contextual factor reward count then
3          collect data points for hypothesized contextual factors
4      if reward count = pre-contextual factor reward count then
5          test hypothesized contextual factor using pearson correlation
6          if contextual factors are found then
7              update set I
8              reinitialize all variables in Algorithm 1
9      collect data points for set I
10     construct and return feature vector
```

Contextual Factor Detection and Context Integration. Algorithm 3 shows our solution to requirement *R4* – the context integration mechanism. The algorithm starts in the pre-contextual factor phase (P1A). In this phase, contextual feature vectors are constructed using information available with the user requests (e.g., age group). Hypothesized contextual factors are observed and stored by the controller for future analysis. When the pre-contextual factor reward threshold is reached, the correlation between the hypothesized contextual factors and process performance is analysed. If the correlation is above a pre-determined threshold, the algorithm state is reset to accommodate new contextual information. This marks the beginning of the post-contextual factor experimentation phase (P1B). From this point onward, contextual feature vectors are constructed using the information from user requests and the observed values of the contextual factors. Finally, when the experimentation request count is achieved, the algorithm switches to the post-experimentation phase (P2). In this phase, the algorithm stops learning from new requests. However, to account for long delays between process instantiation and reward observation, the algorithm continues learning from the requests made in phase P1B.

In summary, we address requirements R1 and R2 through asynchronous partial reward updates using effective and ideal vectors. Requirement R3 is addressed by maintaining a user defined reward ratio τ between complete and incomplete rewards, and handling updates accordingly. Finally, R4 is addressed by identifying and integrating contextual factors in contextual feature vectors.

4 Evaluation

In this section, we analyse the behaviour of the approach and specifically the ProcessBandit algorithm in the presence of contextual factors, and in scenarios where an important PPI is available only for a small number of requests. We also evaluate the response times of the algorithm under various infrastructure settings. Finally, we demonstrate the approach using an example process.

The instance router is prototyped using Python and served by Nginx HTTP server[4] and uWSGI application server[5]. We use Redis[6] as the asynchronous task

[4] https://nginx.org/ Accessed 15-06-2017
[5] https://uwsgi-docs.readthedocs.io/en/latest/ Accessed 15-06-2017
[6] https://redis.io/ Accessed 15-06-2017

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-319-91563-0_2

Table 2: PPI configuration.

	Contextual factor f=1				Contextual factor f=2			
	Profit Margin		User Satisfaction		Profit Margin		User Satisfaction	
Context	Version A	Version B	Version A	Version B	Version A	Version B	Version A	Version B
X	9	11	3	2.5	11	9	2.5	3
Y	11	9	2.5	3	9	11	3	2.5

queue and data store. Two worker processes operate on the asynchronous queue. Tasks that require rescheduling are scheduled after 1 second.

4.1 Convergence Characteristics

In the following experiments, we study how ProcessBandit routes requests to process versions and whether the AB tests converge to the best routing policy given the rewards. We consider two *baselines*: a naïve randomized routing algorithm with uniform request distribution, *random-udr*, and LTAvgR [14].

Our experiment setup consists of a simulated BPMS which returns two PPIs, user satisfaction and profit margin, for two process versions. We assume two process versions, A and B, which perform differently based on the context, X and Y, and an contextual factor *f*. Table 2 summarizes the PPIs returned by each version under various conditions. These PPIs are mapped to the reward design models shown in Fig. 2. p_{ideal} represents user satisfaction score of 5 and profit margin of 20%. p_{ref} represents user satisfaction score of 5 and profit margin of 10%. We chose an optimistic reference point with the philosophy that users who do not provide satisfaction scores are happy, and that profit margin is good. Rewards are derived using the classification model in Fig. 2 with weight mapping of $\{C_1 \mapsto w_1, C_2 \mapsto w_2, \ldots, C_5 \mapsto w_5\}$ such that $w_1 = -1, w_2 = -2, \ldots, w_5 = -5$.

We define the following key terms that we use in the experiments below:
$t_{\mathbf{ppi1}}$**:** the time between request invocation and observation of the first PPI,
$t_{\mathbf{obs}}$**:** the time between request invocation and observation of the full reward,
d**:** delay between the first and the second PPI such that $t_{\text{obs}} = t_{\text{ppi1}} + d$,
ρ**:** ratio between the average request inter-arrival rate and t_{obs}.

Convergence is shown by evaluating *regret* over time. Regret is defined as the difference between the sum of rewards associated with the optimal solution and the sum of rewards collected by pulling the chosen arm [19]. The objective of our algorithm is to find a configuration where the average regret of future actions tends to zero. Graphically, this is the case when the cumulative regret curve tends to become parallel to the x-axis. Given the initial uncertainty about the performance of the versions, the algorithm needs to start with experimentation, and hence by necessity accumulate some regret at first.

Overall behavior. Figure 3 shows cumulative regret of the algorithms using various probabilistic decay functions such that $M = 500$, $d = 0.2 \cdot t_{\text{ppi1}}$ and $f = 1$ for all process instances. In this experiment, we emulate business processes by sampling the completion time of each version from the process execution data of one of the processes from the BPIC 2015 Challenge [18]. The best routing policy

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-319-91563-0_2

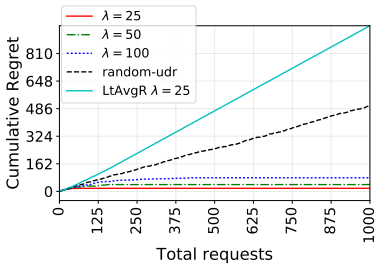

Fig. 3: Cumulative Regret of various algorithms with $d = 0.2 \cdot t_{\text{ppi1}}$.

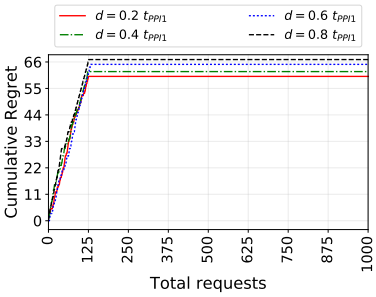

Fig. 4: Convergence at various reward delays.

can be found only if contextual factor *f*, context information, and both PPIs are available. To ensure that these algorithms can be compared, regret for LtAvgR and random-udr is calculated using the weights of reward classes in the same manner as ProcessBandit. We observe that ProcessBandit correctly distributes requests to better performing versions, and converges to the best routing policy. However, LtAvgR consistently makes the wrong decision because it never sees the actual value of the second PPI, and is incapable of updating past rewards.

Figure 4 shows the cumulative regret of ProcessBandit with various delays between the observation times of the first and second PPI. In this experiment, we use deterministic completion times for each process instance so that the value of the PPI delay is the same for all observations. We can observe that the regret curves have similar characteristics, and that the algorithm converges to the best routing policy in all cases. There are some small differences in cumulative regret in all scenarios. However, these differences do no support the idea of a conclusive relationship between the delays and overall regret. The magnitude of cumulative regret can be affected by the non-determinism inherent in the algorithm's experimentation phase, and the order in which the PPIs were observed.

Partial rewards and failures. In this section we evaluate convergence characteristics of ProcessBandit when only one PPI can be observed for some instances. We use an experiment setup with $f = 1, \rho = 160,\ d = 0.2 \cdot t_{\text{ppi1}}$, and a constant execution time for all process instances. Each process instance returns the first PPI (profit margin) immediately after execution. The other PPI (user satisfaction) is either never returned, or returned after a delay – which can be expected if, e.g., users are asked to participate in a short survey.

We define p as the percentage of process instances that return both PPIs. We compare regret characteristics of ProcessBandit with *random-udr* because *random-udr* is agnostic to p. Figure 5 shows the convergence characteristics for parameter values of λ=100, M=500, and τ=0, respectively τ=1. It depicts behavior for values of p that highlight when convergence happens (e.g., 30% for τ=1) and when not (20% in the same configuration). We observe that by increasing τ from 0 to 1, the algorithm can converge when p is smaller.

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-319-91563-0_2

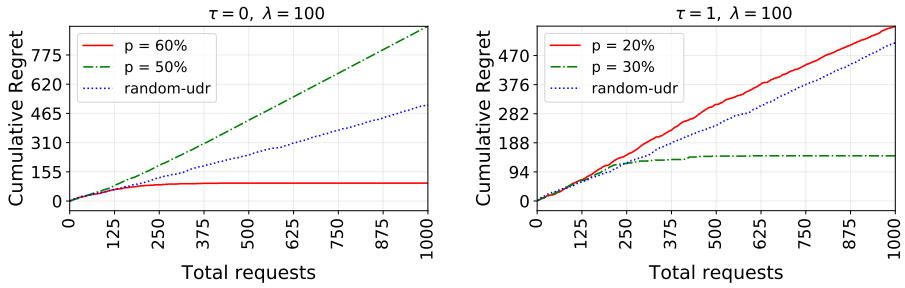

Fig. 5: Convergence with various reward ratio and experiment phase parameters.

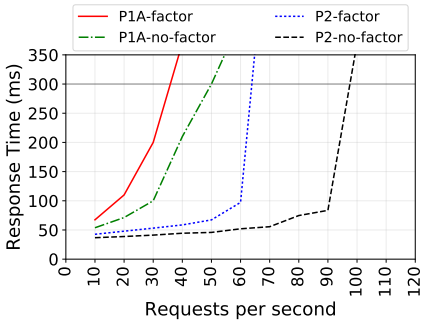

Fig. 6: Response times of various stages and configurations of ProcessBandit.

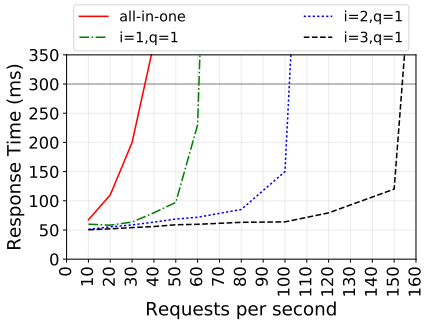

Fig. 7: Response time in Phase P1A. i and q are number of servers for instance router and task queue respectively.

The resilience to partial rewards depends on the values of λ, τ and M. There must be enough *complete* observations in every context so that it is possible to reach a state where the current reward ratio is equal or below τ. In some cases where the best routing policy is found, the algorithm can temporarily perform worse than *random-udr*. Increasing λ to 500 (not shown in figures), convergence is achieved with $p = 20\%$. This is further improved to $p = 10\%$ when τ is increased to 1.5, and finally to $p = 2.5\%$ when M is increased to 750.

4.2 Response Time

We measure end-to-end response time and throughput using two servers, one for the instance router and one for the BPMS. We define our SLA metrics in terms of response time and correctness: our system adheres to SLA if it serves 100% of requests under 300ms. We host these components on Amazon EC2 M4 large instances with 2 vCPUs and 8 GB RAM. We use the reward setup described in Table 2. Figure 6 shows the response times of these configurations, each named with the convention *Phase-Configuration*.

Response times are shown as the average of all requests during a 5 minute burst of the corresponding workload. We observe that the throughput is lower

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-319-91563-0_2

Table 3: User satisfaction model

Outcome	Duration	Version	Satisfaction	
			Ret.	New
Approved	⩽5 wks	A	4	5
		B	5	5
	⩾5 wks	A	3	4
		B	4	4
Rejected	⩽5 wks	A	2	3
		B	3	3
	⩾5 wks	A	1	2
		B	2	2

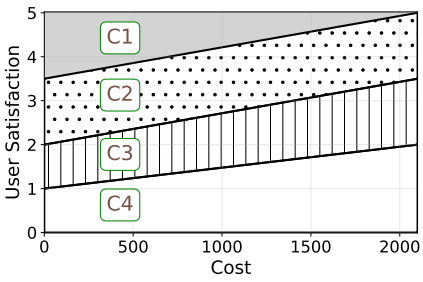

Fig. 8: Performance Classification

and the responses are slower when contextual factor detection is enabled. This is caused by the instance router making additional request to the BPMS to observe the value of *f*. Performance can be improved by adjusting sampling rates of *f*. For example, factors like weather condition can be sampled every minute instead.

We observe that the CPU utilization is generally high (above 90 percent) and proportional to the workload but memory utilization is low (5-7 percent). The *random-udr* algorithm serves up to 400 requests per second under our SLA criterion. On the same infrastructure, *ProcessBandit* achieves a throughput that is between 10% and 25% percent of *random-udr*, depending on the configuration.

Based on this observation, we conduct a second experiment to test the horizontal scalability of *ProcessBandit* at its slowest configuration. This configuration, deployed on a single machine, serves a baseline. Then we deploy the instance router and the asynchronous task queue in separate servers, and horizontally scale the number of the instance router servers. We evaluate response-time and throughput for three deployment configurations. The results are shown in Fig. 7. Because instance router was the bottleneck, we observe that increasing the number of instance router servers increases the throughput.

4.3 AB Test with Synthetic Process

We demonstrate our approach using process versions from the domain of helicopter pilot licensing, as introduced in [14]. The process consists of six activities: Schedule, Eligibility Test, Medical Test, Theory Test, Practical Test, and License Processing. We here add two contextual factors associated with an applicant – age group and applicant type (new or returning). The probability of success in the Medical Test activity is set to be higher for younger age groups. For other activities, success probabilities are the same regardless of age groups.

Activities in Version A of the process are ordered sequentially such that a scheduling activity occurs before each test activity. In Version B, one scheduling activity is performed at the start, which determines the schedules of all the tests, thus reducing the costs of having multiple scheduling activities. We use the activity costs and durations outlined in [14]. Using these process versions, we design an experiment where the process performance is determined by two PPIs:

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-319-91563-0_2

Table 4: Performance of versions A and B

Age Group	Applicant	Reward Class Distribution		Samples	
		Version A	Version B	Version A	Version B
18-24	Returning	C1 - 0% C2 - 60.0% C3 - 40.0% Avg Reward = -2.4	C1 - 59.17% C2 - 11.42% C3 - 29.41% Avg Reward = **-1.7**	29	**304**
	New	C1 - 52.67% C2 - 11.03% C3 - 36.3% Avg Reward = **-1.84**	C1 - 47.06% C2 - 5.88% C3 - 47.06% Avg Reward = -2.0	**299**	34
25-40	Returning	C1 - 0% C2 - 28.57% C3 - 71.43% Avg Reward = -2.71	C1 - 56.16% C2 - 11.64% C3 - 32.19% Avg Reward = **-1.76**	27	**307**
	New	C1 - 61.48% C2 - 14.84% C3 - 23.67% Avg Reward = **-1.62**	C1 - 50.0% C2 - 12.5% C3 - 37.5% Avg Reward = -1.88	**299**	34
40+	Returning	C1 - 0% C2 - 57.14% C3 - 42.86% Avg Reward = -2.43	C1 - 56.14% C2 - 14.04% C3 - 29.82% Avg Reward = **-1.74**	24	**310**
	New	C1 - 50.88% C2 - 21.05% C3 - 28.07% Avg Reward = **-1.77**	C1 - 42.86% C2 - 14.29% C3 - 42.86% Avg Reward = -2.0	**300**	33

satisfaction score, and cost. Rewards are derived from four categories shown in Fig. 8. Satisfaction scores are derived from the outcome and the duration of the process. Satisfaction score is high if the license is approved and processing is fast, and low otherwise. Satisfaction scores also depend on whether the applicant is new or returning – we assume that returning applicants are harsher on the older version. This is shown in Table 3. While the age group is treated as known context, applicant type is treated as a hypothetical contextual factor.

To simulate a scenario where the satisfaction score is not always available, we assume that satisfaction scores are collected within 60 days after process completion. Applicants are notified four times after process completion – on the 7^{th}, 14^{th}, 21^{st}, and 42^{nd} day. The cumulative probability of response is assumed to jump after a notification, a behavior similar to the response rates of web-based career survey [15]. Response probabilities are shown in Fig. 9.

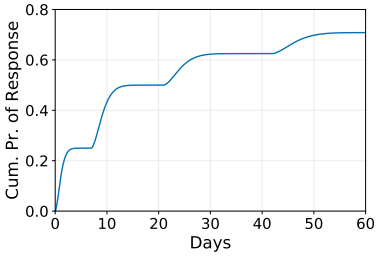

Fig. 9: Probability of receiving satisfaction score over time

With this setup, the algorithm needs to account for two PPIs *(R1)*, the delay in receiving satisfaction scores *(R2)*, availability of satisfaction scores *(R3)*, and the effect of applicant type on the PPIs *(R4)*. The results of performing AB tests

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-319-91563-0_2

on this setup are shown in Table 4. We observe that in all cases more requests are sent to the version that performs better on average (shown in bold).

5 Discussion & Conclusion

Summary We introduce *ProcessBandit*, a dynamic process instance routing algorithm that learns a routing policy based on process performance. The algorithm is supported by a modular architecture. ProcessBandit meets all of our requirements and, while not very fast, can be scaled horizontally. It makes sound decisions in scenarios where performance is determined by delayed PPIs which may be fully observed only for some process instances. It also identifies contextual factors at runtime and uses these factors to make routing decisions.

Discussion. ProcessBandit assumes that overall performance can be summarized using the mean. Rewards are averaged per context and version, which helps the algorithm learn its routing policy. As shown in Table 4, the rewards are not always normally distributed. If other statistical properties are important in decision-making, the mechanism for estimating rewards should be changed in the algorithm. Average performance can be ineffective in scenarios where performance deviations from the norm are small but very important. For example, a tiny number of instances may take exceptionally long time. Rewards for such cases are received late and have negligent effect on the mean. Such cases can be handled by an upper bound on the duration: if the process does not complete within an acceptable time, a strong negative reward can be assigned.

There is a threat to external validity by using synthetic datasets. In previous work [14], we demonstrated that AB-BPM can work on real-world datasets, by deriving a single PPI from the available data. Despite our best efforts, we could not find or produce a real-world dataset that combined all features (multiple PPIs, context, etc.) to evaluate ProcessBandit. We aimed to minimize this risk by producing synthetic datasets based on parameters taken from the literature [15], industry (described in [14]), and a BPI Challenge [18] where possible.

Our contextual factor detection approach is based on correlation. In our experiments, we set the correlation thresholds low, so that the context sensitivity could be evaluated. The detection of contextual factors does not need to be based on correlation. Our approach is not tied to how these contextual factors are identified. Contextual factor detection is however a challenge per se that needs further investigation.

Conclusion and Future Work. Unlike prior work on AB testing, our solution provides a risk-managed approach tailored for the requirements of business processes. We demonstrate that this solution meets these requirements by evaluating the behaviour of the routing algorithm, the horizontal scalability of the approach, and its effectiveness in a synthetic business process. Our future plans include extension of the approach to accommodate other statistical properties in reward evaluation and upper bound on duration. We also plan to collaborate with domain experts to conduct field tests in the industry.

Pre-print copy of the manuscript published by Springer
(available at link.springer.com)
identified by DOI: 10.1007/978-3-319-91563-0_2

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

References

1. van der Aalst, W.M.P., Rosemann, M., Dumas, M.: Deadline-based escalation in process-aware information systems. Decis. Support Syst. 43(2), 492–511 (2007)
2. Agrawal, S., Goyal, N.: Thompson sampling for contextual bandits with linear payoffs. In: Intl. Conf. Machine Learning, ICML (2013)
3. Branke, J., Deb, K., Miettinen, K., Slowinski, R. (eds.): Multiobjective Optimization, Interactive and Evolutionary Approaches [outcome of Dagstuhl seminars]., Lecture Notes in Computer Science, vol. 5252. Springer (2008)
4. Burtini, G., Loeppky, J., Lawrence, R.: A survey of online experiment design with the stochastic multi-armed bandit. CoRR abs/1510.00757 (2015)
5. Chu, W., Li, L., Reyzin, L., Schapire, R.E.: Contextual bandits with linear payoff functions. In: Intl. Conf. on Artificial Intelligence and Statistics. pp. 208–214 (2011)
6. Crook, T., Frasca, B., Kohavi, R., Longbotham, R.: Seven pitfalls to avoid when running controlled experiments on the web. In: ACM SIGKDD. pp. 1105–1114 (2009)
7. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer (2013)
8. He, H., Ma, Y.: Imbalanced learning: foundations, algorithms, and applications. John Wiley & Sons (2013)
9. Holland, C.W.: Breakthrough Business Results With MVT: A Fast, Cost-Free "Secret Weapon" for Boosting Sales, Cutting Expenses, and Improving Any Business Process. John Wiley & Sons (2005)
10. 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)
11. 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)
12. Li, L., Chu, W., Langford, J., Schapire, R.E.: A contextual-bandit approach to personalized news article recommendation. In: Intl. Conf. World Wide Web (2010)
13. 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)
14. Satyal, S., Weber, I., Paik, H., Di Ciccio, C., Mendling, J.: AB-BPM: performance-driven instance routing for business process improvement. In: BPM (2017)
15. Sauermann, H., Roach, M.: Increasing web survey response rates in innovation research: An experimental study of static and dynamic contact design features. Research Policy 42(1), 273–286 (2013)
16. Silver, D., Newnham, L., Barker, D., Weller, S., McFall, J.: Concurrent reinforcement learning from customer interactions. In: ICML. pp. 924–932 (2013)
17. Sutton, R.S., Barto, A.G.: Introduction to Reinforcement Learning. MIT Press, Cambridge, MA, USA, 1st edn. (1998)
18. Teinemaa, I., Leontjeva, A., Masing, K.O.: BPIC 2015: Diagnostics of building permit application process in dutch municipalities. BPI Challenge Report 72 (2015)
19. Vermorel, J., Mohri, M.: Multi-armed bandit algorithms and empirical evaluation. In: Proc. ECML European Conference on Machine Learning. pp. 437–448 (2005)

Pre-print copy of the manuscript published by Springer
(available at `link.springer.com`)
identified by DOI: `10.1007/978-3-319-91563-0_2`

This document is a pre-print copy of the manuscript
(Satyal et al. 2018)
published by Springer
(available at link.springer.com).

The final version of the paper is identified by DOI: 10.1007/978-3-319-91563-0_2

References

Satyal, Suhrid, Ingo Weber, Hye-young Paik, Claudio Di Ciccio, and Jan Mendling (2018). "AB Testing for Process Versions with Contextual Multi-armed Bandit Algorithms". In: *CAiSE*. Ed. by John Krogstie and Hajo A. Reijers. Vol. 10816. Lecture Notes in Computer Science. Springer, pp. 19–34. ISBN: 978-3-319-91562-3. DOI: 10.1007/978-3-319-91563-0_2.

BibTeX

```
@InProceedings{   Satyal.etal/CAiSE2018:ABTestingforProcessVersions,
  author        = {Satyal, Suhrid and Weber, Ingo and Paik, Hye{-}young and
                   Di Ciccio, Claudio and Mendling, Jan},
  title         = {{AB} Testing for Process Versions with Contextual
                   Multi-armed Bandit Algorithms},
  booktitle     = {CAiSE},
  year          = {2018},
  pages         = {19--34},
  crossref      = {CAiSE2018},
  doi           = {10.1007/978-3-319-91563-0_2},
  keywords      = {Multi-armed bandit; Business Process Management; AB
                   Testing; Process Performance Indicators}
}
@Proceedings{     CAiSE2018,
  title         = {Advanced Information Systems Engineering - 30th
                   International Conference, CAiSE 2018, Tallinn, Estonia,
                   June 11-15, 2018, Proceedings},
  year          = {2018},
  editor        = {John Krogstie and Hajo A. Reijers},
  volume        = {10816},
  series        = {Lecture Notes in Computer Science},
  publisher     = {Springer},
  isbn          = {978-3-319-91562-3},
  doi           = {10.1007/978-3-319-91563-0}
}
```