Deep Reinforcement Learning for Resource Allocation in Business Processes

Kamil Żbikowski, Michał Ostapowicz, Piotr Gawrysiak

Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warsaw, Poland
kamil.zbikowski@pw.edu.pl,michal.ostapowicz@pw.edu.pl,piotr.gawrysiak@pw.edu.pl

Abstract. Assigning resources in business processes execution is a repetitive task that can be effectively automated. However, different automation methods may give varying results that may not be optimal. Proper resource allocation is crucial as it may lead to significant cost reductions or increased effectiveness that results in increased revenues.
In this work, we first propose a novel representation that allows the modeling of a multi-process environment with different process-based rewards. These processes can share resources that differ in their eligibility. Then, we use double deep reinforcement learning to look for an optimal resource allocation policy. We compare those results with two popular strategies that are widely used in the industry. Learning optimal policy through reinforcement learning requires frequent interactions with the environment, so we also designed and developed a simulation engine that can mimic real-world processes.
The results obtained are promising. Deep reinforcement learning based resource allocation achieved significantly better results compared to two commonly used techniques.

Keywords: resource allocation · deep reinforcement learning · Double DQN · process optimization

1 Introduction

In process science, there is a wide range of approaches that are employed in different stages of operational processes' life cycles. Following [1], these include, among others, optimization and stochastic techniques. Business processes can be also categorized according to the following perspectives: control-flow, organizational, data, and time perspective [2]. Resource allocation is focused on the organizational perspective utilizing optimization and stochastic approaches.

As it was emphasized in [3] resource allocation, while being important from the perspective of processes improvement, did not receive much attention at the time. However, as it was demonstrated in [4] the problem received much more attention in the last decade, which was reflected in the number of published scientific papers.

This paper addresses the problem of resource allocation with the use of methods known as approximate reinforcement learning. We specifically applied recent

advancements in deep reinforcement learning such as double deep q-networks (double DQN) described in [5]. To use those methods we firstly propose a representation of a business processes suite that helps to design the architecture of neural networks in terms of appropriate inputs and outputs.

To the best of our knowledge, this is the first work that proposes a method utilizing double deep reinforcement learning for an on-line resource allocation for a multiple-process and multi-resource environment. Previous approaches either used so-called "post mortem" data in the form of event logs (e.g. [6]), or applied on-line learning, but due to the usage of tabular algorithms were limited by the exploding computational complexity when the number of possible states increased.

In the next section, we provide an overview of reinforcement learning methods and outline improvements of deep learning approaches over existing solutions. Then we analyze and discuss different approaches to resources allocation. In Section 3 we outline our approach for modeling operational processes for the purpose of training resource allocation agents. In Section 4 we describe the simulation engine used in training and its experimental setup. In Section 5 we evaluate the proposed approach and present outcomes of the experiments. In Section 6 we summarize the results and sketch potential future research directions.

2 Background and Related Work

2.1 Deep Reinforcement Learning

Following [7], reinforcement learning is "learning what to do – how to map situations to actions – so as to maximize a numerical reward signal". There are two main branches of reinforcement learning, namely tabular and approximate methods. The former provide a consistent theoretical framework that under certain conditions guarantees convergence. Their disadvantage is increasing computational complexity and memory requirements when the number of states grows. The latter are able to generalize over a large number of states but do not provide any guarantee of convergence.

The methods that we use in this work find optimal actions indirectly, identifying optimal action values for each state-action pair. Following recursive Bellman equation for the state-action pair [7], where $p(s', r|s, a)$ is a conditional probability of moving to state s' and receiving reward r after taking action a in state s; $\pi_(a|s)$ is the probability of taking action a in state s; $\gamma \in [0, 1]$ is a discount factor:

$$q_\pi(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \sum_{a'} \pi(a'|s')q_\pi(s', a')], \tag{1}$$

an optimal policy is a policy that at each subsequent step takes an action that maximizes state-action value, that is $q_*(s, a) = max_\pi q_\pi(s, a)$.

When we analyze equation 1 we can intuitively understand problems with iterative tabular methods for finding optimal policy π^* for high-dimensional state spaces. Fortunately, recent advancements in deep learning methods allow

for further enhancement of approximate reinforcement learning methods with a most visible example being human-level results for Atari suite [8] obtained with the use of double deep Q-network [9].

2.2 Resource Allocation

In [4] we can find a survey of human resource allocation methods. The spectrum of approaches is wide. In [10], [11], [12], [13] and [14] we can find solutions based on static, rule based algorithms.

There is a number of approaches for resource allocation that rely on applying predictive models. In [15] an offline prediction model based on LSTM is combined with extended minimum cost and maximum flow algorithms.

In [16] authors introduce Reinforcement Learning Based Resource Allocation Mechanism that utilizes Q-learning for the purpose of resource allocation. For handling multiple business processes, the queuing mechanism is applied.

Reinforcement learning has been also used for the task of proactive business process adaptation [17] [18]. The goal there is to monitor the particular business process case while it is running and intervene in case of any detected upcoming problems.

The evaluations conducted in aforementioned works are either based on simulations ([18], [16]) or on analysis of historical data, mostly from Business Process Intelligence Challenge ([17], [15], [19]). The latter has an obvious advantage of being real-world based dataset while simultaneously being limited by the number of available cases. The former offers a potentially infinite number of cases, but alignment between simulated data and real business processes is hard to achieve.

In [20] authors proposed a deep reinforcement learning method for business process optimization. However, their research objective is concentrated on analyzing which parameters of DQN are optimal.

3 Approach

This section describes the methods that we used to conduct the experiment. First, we will introduce concepts related to business process resource allocation. Then we will present double deep reinforcement learning [21] for finding optimal resource allocation policy. By optimal resource allocation policy, we mean such that maximizes the number of completed business process cases in a given period.

As it was pointed out earlier, both tabular and approximate algorithms in the area of reinforcement learning require frequent interaction with the execution environment. For the purpose of this work, we designed and developed a dedicated simulation environment that we call Simulation Engine. However, it can serve as a general-purpose framework for testing resource allocation algorithms as well. Concepts that we use for defining the business process environment assume the existence of such an engine. They incorporate parameters describing the level of uncertainty regarding their instances. The purpose here is to replicate stochastic behavior during process execution in real-world scenarios.

We imagine a business process workflow as a sequence of tasks[1] that are drawn from the queue and are being executed by adequate resources (both human and non-human). Each task realization is in fact an instance of a task specification described below. The task here is considered as an unbreakable unit of work that a resource can be assigned to and works on for a specified amount of time.

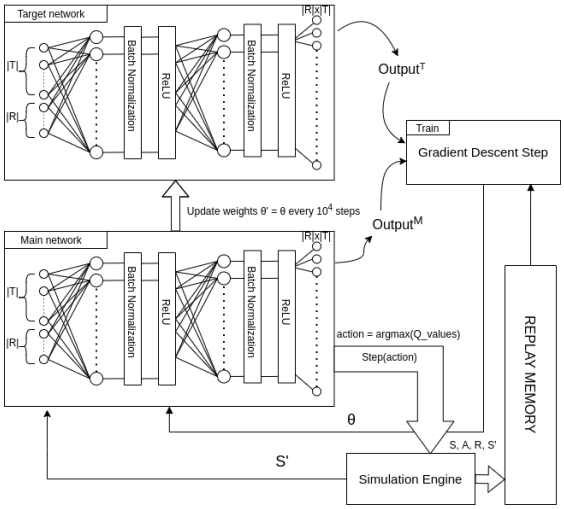


Fig. 1: Training architecture diagram. The learning process is centered around Simulation Engine that takes action from the main network and returns the reward and the next state. The architecture above follows the double deep Q-network (DDQN) approach [21].

Definition 1 (Task). *Let the tuple* $(i, \mathcal{C}^i, \mathit{d}, s, \mathit{b})$ *define a task* t_i *that is a single work unit represented in the business process environment where:*

- i *is a unique task identifier where* $i \in \{0, 1, 2, ...\}$,
- $\mathcal{C}^i$ *is a set of transitions from a given task* i,
- $\mathit{d} \in \mathbb{R}^+$ *is a mean task duration with* s *being its standard deviation and*
- $\mathit{b} \in \{0, 1\}$ *indicates whether it is a starting task for particular business process.*

Each task in the business process (see e.g. Figure 2a) may have zero or more connections from itself to other tasks.

[1] Task here should not be confused with the task definition used in reinforcement learning literature where it actually means the objective of the whole learning process. In the RL sense, our task would be to "solve" Business Process Suite (meaning obtaining as much cumulative reward as possible) in the form of Definition 6.

Definition 2 (Task Transition). *For a given task t_i a task transition c_j^i is a tuple (j, p) where j is an unique identifier of a task that this transitions refers to and p is a probability of this transition. If $i = j$ it is a transition to itself.*

Definition 3 (Resource). *Let the tuple (k) define a single resource r_k where $k \in \{0, 1, 2, ...\}$ is a unique resources identifier. To refer to the set of all resources, we use $\hat{\mathcal{R}}$.*

Definition 4 (Resource Eligibility). *If a resource r_k can be assigned to a task t_i it is said it is eligible for this task. Set $\mathcal{E}^i = \{e_k^i : e_k^i \in R^+\}$ contains all resource eligibility modifiers for a given task i. The lower the e_k^i, the shorter is the expected execution of task t_i. To refer to the set of all properties of eligibility for all defined resources $\hat{\mathcal{R}}$ we use $\hat{\mathcal{E}}$.*

The expected execution time of a task t_i is calculated by multiplying its duration by the resource eligibility modifier e_k^i.

Definition 5 (Business Process). *Let a tuple $(m, f_m, \mathbb{R}_m, \mathcal{T}_m)$ define a business process $\mathcal{P}_m$ where m is a unique identifier of a process $\mathcal{P}_m$ and $\mathcal{T}_m$ is a set of tasks belonging to the process $\mathcal{P}_m$ and $t_i \in \mathcal{T}_m \implies \neg\exists n : n \neq m \wedge t_i \in \mathcal{T}_n$. The relative frequency of a particular business process is defined by f_m. By $\mathbb{R}_m$ we refer to the reward that is received by finishing this business process instance. To refer to the set of all defined business processes, we use $\hat{\mathcal{P}}$.*

An example of a business process can be found in Figure 2a. Nodes represent tasks and their identifiers. Arrows define possible task transitions from particular nodes. The numbers on the arrows represent transition probabilities to other tasks.

Definition 6 (Business Process Suite). *Let a tuple $(\hat{\mathcal{R}}, \hat{\mathcal{E}}, \hat{\mathcal{P}})$ define a Business Process Suite that consists of a resources set $\hat{\mathcal{R}}$, resources eligibility set $\hat{\mathcal{E}}$ and business processes set $\hat{\mathcal{P}}$ such that: $\forall r_k \in \hat{\mathcal{R}}\ \exists m, i\ e_k^i \in \hat{\mathcal{E}} \wedge t_i \in \mathcal{T}_m \wedge \mathcal{P}_m \in \hat{\mathcal{P}}$*

Business Process Suite is a meta definition of the whole business processes execution environment that consists of tasks that aggregate to business processes and resources that can execute tasks in accordance with the defined eligibility. We will refer to the instances of business processes as business process cases.

Definition 7 (Business Process Case). *Let a tuple $(\mathcal{P}_m, i, o)$ define a business process case $\tilde{\mathcal{P}_m}$ where $\mathcal{P}_m$ is a business process definition, i is a current task that is being executed and $o \in \{0, 1\}$ is information whether it is running (0) or was completed (1).*

Definition 8 (Task Instance). *Let a tuple (i, r_k) be a task instance $\tilde{t_i}$. At a particular moment of execution, there exists exactly one task instance matching business process case property i. The exact duration is determined by properties d and s of task definition t_i.*

Definition 9 (Task Queue). *Let the ordered list* $(\mathcal{N}^{t_0}, \mathcal{N}^{t_1}, \mathcal{N}^{t_2}, ..., \mathcal{N}^{t_i})$ *define a task queue that stores information about the number of task instances* $\mathcal{N}^{t_i}$ *for a given task* t_i.

Property 1. Direct consequence of definitions 5, 7, 8 and 9 is that number of task instances in the task queue matching the definition of task with identifiers from particular business processes is equal to the number of business process cases.

The process of learning follows the schema defined in [9] and [5]. We use two sets of weights θ and θ'. The former is used for online learning with random mini-batches sampled from a dedicated experience replay queue $\mathcal{D}$. The latter is updated periodically to the weights of the more frequently changing counterpart. The update period used in tests was 10^4 steps. Detailed algorithm, based on [21], is outlined in listing 1.

Algorithm 1 Double DQN training loop

Initialize number of episodes E, and number of steps in episode M
Initialize batch size β ▷ Set to 32 in tests
Initialize randomly two sets of neural network weights θ and θ'
$\mathcal{D} := \{\}$ ▷ Replay memory of size $E * M * 0.1$
Initialize environment $\mathcal{E}$
for e=0 **in** E **do**
    S := RESET($\mathcal{E}$)
    for m=0 **in** M **do**
        if RANDOM() $< \epsilon$ **then**
            a := SELECTRANDOMACTION()
        else
            a := $argmax_a Q(S, a; \theta)$
        end if
        $\mathbb{S}'$, $\mathbb{R}$:= STEP($\mathcal{E}$, a)
        Put a tuple (S, a, R, S') in $\mathcal{D}$
        Sample β experiences from $\mathcal{D}$ to $(\mathbb{S}, \mathbb{A}, \mathbb{R}, \mathbb{S}')$
        $Q_{target} := \mathbb{R} + \delta * Q(\mathbb{S}', \text{argmax}_a\, Q(\mathbb{S}', a; \theta); \theta')$
        $Q_{current} := Q(S, a; \theta)$
        $\theta_{t+1} = \theta_t + \nabla_{\theta_t}(Q_{target} - Q_{current})^2$
        Each 10^4 steps update $\theta' := \theta$
    end for
end for

In Figure 1 an architecture of a system used in the experiment is presented in accordance with main data flows. It is a direct implementation of the training algorithm described in Algorithm 1. We used two neural networks: main and target. Both had the same architecture consisting of one input layer with $|\mathcal{R}| + |\mathcal{T}|$ inputs, two densely connected hidden layers containing 32 neurons each, and one output layer with $|\mathcal{R}|\chi|\mathcal{T}|$ outputs. After each hidden layer, there is a Batch Normalization layer [22]. Its purpose is to scale each output from hidden neuron

layer before computing the activation function. This operation improves training speed by reducing undesirable effects such as vanishing / exploding gradient updates.

The input configuration we used is defined as follows:

$$\mathbb{S} = [\rho_0, \rho_1, ...\rho_{|\mathcal{R}|-1}, \zeta_0, \zeta_1, ..., \zeta_{|\mathcal{T}|-1}] \tag{2}$$

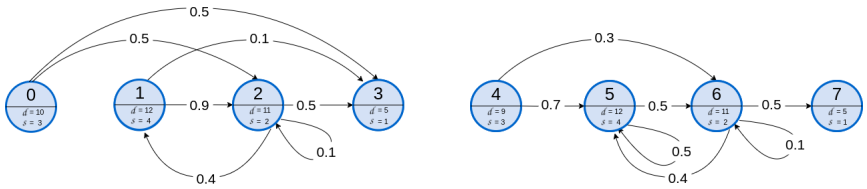
where $\rho_k = i$ refers to the resource assignment to one of its eligible task, and $\zeta_i = \mathcal{N}^{t_i} / \sum_{l=0}^{|\mathcal{T}|-1} \mathcal{N}^{t_l}$ is a relative load of a a given task with respect to all the tasks present in the task queue.

Outputs of the neural network are an approximation of a q-value for each of the available actions. Action here is assigning a particular resource to a particular task or taking no action for a current time step. Thus, number of outputs equals $|\mathcal{R}||\mathcal{T}| + 1$. This number grows quickly with the number of resources and tasks. This, in turn, may lead to a significant increase in training time or even inability to obtain adequate q-value estimation.

In RL there exists a separation between continuing and episodic RL tasks [7]. The former are ending in a terminal state and differ in the rewards for the different outcomes. The latter are running infinitely and accumulate rewards over time. The business processes suite is a continuing RL task in its nature. However, in our work, we artificially terminate each execution after M steps simulating an episodic environment. We observed that it gave much better results than treating the whole set of business processes as a continuing learning task. As it is shown in Section 4 agents trained in such a way can be used in a continuing setup without loss of their performance.

4 Experimental setup

This section briefly describes the setup of the experiments that we have conducted to assess the proposed methods and parametrization of a business process suite used for the evaluation.



(a) Graph of a first business process. (b) Graph of a second business process.

Fig. 2: Business processes used in the evaluation.

To evaluate the proposed method we devised a business processes suite containing two business processes $m = 0$ and $m = 1$. Although they are quite small

in terms of the number of tasks, the tasks transitions are nondeterministic which intuitively makes the learning process harder.

In Figures 2a and 2b we can see both processes' graphs along with information about their tasks' parametrization. In Table 1 we can see available resources from the testing suite along with the information about their eligibility in regard to particular tasks.

Both processes have the same reward $\mathbb{R}_0 = \mathbb{R}_1 = 1$ which is received for each completed business process case. They differ in their relative frequency, which for first process is $f_0 = 1$ and $f_1 = 6$ for the second one.

The resources we use in our experimental setup are of the same type, differing only in their eligibility in regard to the tasks.

	Resources		
Task ID	0	1	2
0	-	0.75	2.8
1	1.4	0.3	-
2	0.3	-	2.7
3	-	2.7	0.1
4	0.6	2.6	-
5	0.4	-	10.5
6	1.1	-	1.7
7	0.4	0.6	2.5

Table 1: Resource eligibility. Values in cells define resource efficiency that is used in Simulation Engine. Final duration is obtained by multiplying duration d of a particular task by the adequate value from the table. A lack of value indicates that a particular resource is not eligible for a given task.

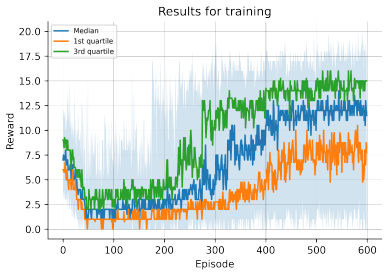
In terms of algorithm parametrization, we set the number of episodes E to 600 and the number of steps in a single episode to 400. ϵ according to [5] was linearly annealed from 1 to 0.1 over first $E*M*0.1$ steps. The size of the memory buffer was set to $E * M * 0.1$ elements.

5 Results and discussion

We run 30 tests for the test suite. The results are presented in Figure 3a. We can see that the variance in the cumulative sum of rewards is tremendous. Best models achieve up to 20 units of reward while the worst keep their score around zero.

Our findings are consistent with the general perception of how deep reinforcement learning works [23]. In particular, a training model that achieves satisfactory results strongly depends on weights initialization.

As we can see in Figure 3b the value of a loss function also varies significantly. Moreover, its value after the initial drop steadily increases with subsequent episodes. This is a phenomenon that is characteristic of DQN. The error measures the difference between training and main network outputs. This value



(a) Sum of rewards for subsequent episodes

(b) Training loss.

Fig. 3: Training on the test suite over 30 training runs.

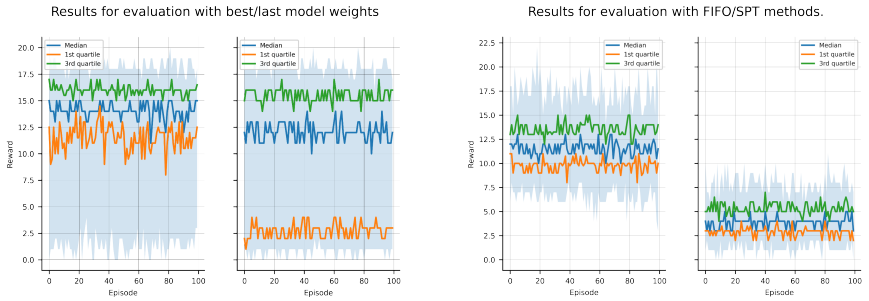
is not directly connected with the optimization target - maximizing the cumulative reward over all steps.

In [5] authors recommend saving model parameters if they are better than the best previously seen (in terms of cumulative reward) during the current training run. This approach allows addressing - to some extent - a catastrophic forgetting effect and overall instability of approximate methods. For each run we save both the best and last episode's weights. After the training phase, we got 30 models as a result of keeping parameters giving the highest rewards during learning and 30 models with parameters obtained at the end of training. The distribution over all runs can be seen in Figure 4a. We can see that the models with the best parameters achieve significantly higher cumulative rewards. The median averaged over 100 episodes was 14.04 for the best set of parameters and 12.07 for the last set.

To assess the results obtained by the deep learning agent we implemented two commonly used heuristics:

- FIFO (first in, first out) - the first-in-first-out policy was implemented in an attempt to avoid any potential bias while resolving conflicts in resource allocation. In our case, instead of considering task instances themselves, we try to allocate resources to the business process cases that arrived the earliest.
- SPT (shortest processing time) - our implementation of the shortest processing time algorithm tries to allocate resources to the task instances that take the shortest time to complete (without taking into account resource efficiencies for tasks). Thanks to this policy, we are able to prevent the longest tasks from occupying resources when these resources could be used to complete other, much shorter tasks and therefore shorten the task queue.

We conducted the same test lasting 100 episodes for both heuristics. Results are presented in Figure 4b. The median averaged over 100 episodes was 11.54 for FIFO and 3.88 for SPT. SPT results were far below the FIFO. Comparing the results of the best model from the left side of Figure 4a with results for FIFO



(a) Best (left) and last (right) model. (b) FIFO and SPT algorithms

Fig. 4: Results over 30 runs.

from the left side of Figure 4b, we can see that the cumulative reward for deep learning models is larger in the majority of episodes.

The improvement achieved by the deep RL model with each episode lasting 400 steps is not large considering its absolute value. The median FIFO agent's reward oscillates around 11, while the median deep RL's around 14. The question that arises here is whether this relation will hold with long (potentially infinitely) lasting episodes? To answer it, we conducted an experiment with 100 episodes with 5000 steps each. The results are presented in Figure 5. We can see that the gap between rewards for DQN model and for FIFO increased. The average episode reward for DQN was 210.52, while for FIFO 145.84 and 80.2 for SPT.

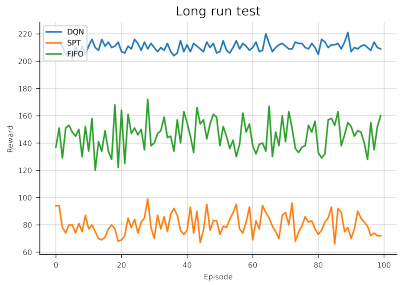


Fig. 5: Long run test for best model achieved during training compared to FIFO and SPT approaches. Each episode lasted 5000 time steps.

6 Conclusions and future work

In this paper, we applied double deep reinforcement learning for the purpose of resource allocation in business processes. Our goal was to simultaneously

optimize resource allocation for multiple processes and resources in the same way as it has to be done in real-world scenarios.

We proposed and implemented a dedicated simulation environment that enables an agent to improve its policy in an iterative manner obtaining information about the next states and rewards. Our environment is thus similar to OpenAI's Gym. We believe that along with processes' definitions, it may serve as a universal testing suite improving the reproducibility of the results for different resource allocation strategies.

We proposed a set of rules for defining business processes suites. They are the formal representation of real-world business process environments.

The results of the double DQN algorithm for resources allocation were compared with two strategies based on common heuristics: FIFO and SPT. The deep RL approach obtained results that are 44% better than FIFO and 162% better than SPT. We were not able to directly compare our results to previously published studies as they are relatively hard to reproduce. This was one of the main reasons to publish the code of both our simulation engine and training algorithm. We can see this as a first step toward a common platform that will allow different resource allocation methods to be reliably compared and assessed.

As for future work, it would be very interesting to train a resource allocation agent for a business process suite with a larger number of business processes that would be more deterministic compared to those used in this study. Such a setup would put some light on a source of complexity in the training process.

The number of potential actions and neural networks' outputs is a significant obstacle in applying the proposed method for complex business process suites with many processes and resources. In our future work, we plan to investigate other deep reinforcement learning approaches, such as proximal policy optimization, which tend to be more sample efficient than standard double DQN.

Reproducibility Source code: https://github.com/kzbikowski/ProcessGym

References

1. Wil Van Der Aalst. Data science in action. In *Process mining*, pages 3–23. Springer, 2016.
2. Wil MP Van der Aalst. Business process management: a comprehensive survey. *International Scholarly Research Notices*, 2013, 2013.
3. Zhengxing Huang, Xudong Lu, and Huilong Duan. Mining association rules to support resource allocation in business process management. *Expert Systems with Applications*, 38(8):9483–9490, 2011.
4. Michael Arias, Rodrigo Saavedra, Maira R Marques, Jorge Munoz-Gama, and Marcos Sepúlveda. Human resource allocation in business process management and process mining. *Management Decision*, 2018.
5. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.
6. Tingyu Liu, Yalong Cheng, and Zhonghua Ni. Mining event logs to support workflow resource allocation. *Knowledge-Based Systems*, 35:320–331, 2012.

7. Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
8. Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.
9. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
10. Zhengxing Huang, Xudong Lu, and Huilong Duan. Resource behavior measure and application in business process management. *Expert Systems with Applications*, 39(7):6458–6468, 2012.
11. Weidong Zhao, Liu Yang, Haitao Liu, and Ran Wu. The optimization of resource allocation based on process mining. In *International Conference on Intelligent Computing*, pages 341–353. Springer, 2015.
12. Michael Arias, Eric Rojas, Jorge Munoz-Gama, and Marcos Sepúlveda. A framework for recommending resource allocation based on process mining. In *International Conference on Business Process Management*, pages 458–470. Springer, 2016.
13. Giray Havur, Cristina Cabanillas, Jan Mendling, and Axel Polleres. Resource allocation with dependencies in business process management systems. In *International Conference on Business Process Management*, pages 3–19. Springer, 2016.
14. Jiajie Xu, Chengfei Liu, and Xiaohui Zhao. Resource allocation vs. business process improvement: How they impact on each other. In *International Conference on Business Process Management*, pages 228–243. Springer, 2008.
15. Gyunam Park and Minseok Song. Prediction-based resource allocation using lstm and minimum cost and maximum flow algorithm. In *2019 International Conference on Process Mining (ICPM)*, pages 121–128. IEEE, 2019.
16. Zhengxing Huang, Wil MP van der Aalst, Xudong Lu, and Huilong Duan. Reinforcement learning based resource allocation in business process management. *Data & Knowledge Engineering*, 70(1):127–145, 2011.
17. Andreas Metzger, Tristan Kley, and Alexander Palm. Triggering proactive business process adaptations via online reinforcement learning. In *International Conference on Business Process Management*, pages 273–290. Springer, 2020.
18. Zhengxing Huang, Wil MP van der Aalst, Xudong Lu, and Huilong Duan. An adaptive work distribution mechanism based on reinforcement learning. *Expert Systems with Applications*, 37(12):7533–7541, 2010.
19. Alexander Palm, Andreas Metzger, and Klaus Pohl. Online reinforcement learning for self-adaptive information systems. In *International Conference on Advanced Information Systems Engineering*, pages 169–184. Springer, 2020.
20. Johan Silvander. Business process optimization with reinforcement learning. In *International Symposium on Business Modeling and Software Design*, pages 203–212. Springer, 2019.
21. Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.
22. Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.
23. Alex Irpan. Deep reinforcement learning doesn't work yet. https://www.alexirpan.com/2018/02/14/rl-hard.html, 2018.