Online Resource Allocation to Process Tasks Under Uncertain Resource Availabilities

Michel Kunkler, Stefanie Rinderle-Ma
TUM School of Computation, Information and Technology
Technical University of Munich
Garching, Germany
{michel.kunkler,stefanie.rinderle-ma}@tum.de

***Abstract*—Allocating resources to process tasks during runtime (online) is hard. A solution method for such allocation is required to be computationally efficient while being subjected to uncertainties such as resources suddenly becoming (un)available. Resource allocation problems where task processing times differ across resources can be formalized as an assignment or parallel machines scheduling problem. This work presents adaptations to both problem formulations to address resource (un)availabilities. These adaptations require a prediction model to estimate the processing time of a task for all of its authorized resources. We evaluate and compare the proposed adaptations with existing allocation approaches on two process simulation models created from an artificial and a real-life event log. Our results show that both approaches can outperform traditional allocation strategies, such as the shortest queue, random, round-robin, and batch-allocation approaches.**

***Index Terms*—Resource Allocation, Process Analytics, Process Monitoring, Resource Unavailabilities**

I. Introduction

Process automation is achieved by coordinating "that all work is done at the right time by the right resource" [1]. To this end, Process-Aware Information Systems (PAISs) either follow the *push-principle*, i.e., the PAIS actively allocates a task to a resource by putting the task into a resource's worklist, or the *pull-principle*, i.e., letting the resource decide which task to execute next [2].

PAISs, which monitor the process execution and available resources within an organization during runtime, can take advantage of its bird's-eye perspective for providing operational support, e.g., by predictions or recommendations. At this, recommendations are similar to predictions but are given with respect to one or multiple objectives, e.g., minimizing cycle times or costs [3]. Providing optimal recommendations with respect to one or multiple objectives can be formalized as a *stochastic (multi-objective) optimization problem*, given that the business processes' inherent uncertainties can be quantified. Formulating and solving such a problem directly can be infeasible in practice for two reasons: Firstly, it is hard to set up an optimization model that precisely resembles reality, e.g., correctly quantifying uncertainties or considering all factors affecting the business objectives. Secondly, because of the computational complexity of stochastic [4] and multi-objective [5] optimization problems. In this work, we assume a high degree of uncertainty regarding resource availabilities. We assume that resources can suddenly become (un)available, resembling, e.g., organizations that have implemented flexitime working policies.

Resource allocation under this assumption is difficult when the expected processing times of tasks differ across resources: Consider a simple scenario where the objective is to minimize the average task completion time with a single idle resource and one active task. Given that the expected processing time of the available resource is overly long for the task, the question is whether to allocate the task immediately or postpone the allocation. Postponing the allocation can be advantageous when another resource that is able to process the task more quickly will soon become available, or conversely, when another task becomes active, which the first resource can process faster than the first task.

In this work, we present two approaches that can decide upon a task's arrival whether to allocate a task immediately or to postpone its allocation: Our two approaches are adaptations of the formulations of two renowned combinatorial optimization problems: the *assignment problem* and the *parallel machines scheduling problem*. The allocation decisions from solving the *assignment problem* are short-sighted, as only the most suitable allocations for a maximum of one task per resource are considered. Conversely, the allocation decisions from the *parallel machines scheduling problem* take the allocations of all tasks to all resources into account, which comes with a higher theoretical computational complexity. For both adaptations, we introduce dummy resources, which are no actual resources but act as placeholders representing the choice to leave a task unallocated when no adequate resource is available.

We prototypically implemented both approaches and evaluated them against each other, and other allocation approaches on two process simulation models discovered from event logs. We executed our implementations on two simulation models that differ in their degree of uncertainty; e.g., they follow different control-flows and have different resource availabilities. We then compared the allocation decisions on three metrics, namely the *average cycle time*, the average time from a process instance's arrival until its completion, the *average resource occupation*, the average proportion of resources being occupied during their presences, and the *weighted resource fairness*, the summed deviation from the mean resource occu-

 DOI: 10.1109/ICPM63005.2024.10723280

pation weighted by the resources actual presences. Our results show that both approaches can outperform other approaches regarding a reduced *average cycle time* and *average resource occupation*.

In Sect. II, we introduce related approaches and discuss similarities and differences to our two approaches. In Sect. III, we formally describe our two resource allocation approaches. In Sect. IV, we compare our approaches against each other and related approaches, while Sect. V concludes this paper.

II. Related Work

Most works concerning resource allocation in Business Process Management Systems (BPMSs) have resorted to simple rule-based allocation strategies or apply heuristics [6]. [2] have identified three simple allocation strategies that have found application in BPMSs, namely Round-Robin allocation (R-RRA), Random allocation (R-RMA), and Shortest Queue First allocation (R-SHQ).

Some works have presented offline scheduling approaches, i.e., all information is known from the beginning: In [7] the control-flow perspective is formalized as an Answer Set Programming (ASP) formulation from which an ASP solver can obtain a schedule. In [8, 9], the control-flow and resource perspective are formalized as a timed Petri net, and solutions are obtained from the reachability graph. However, in these approaches, most properties are assumed to be deterministic: Only [7] consider control-flow induced uncertainty, i.e., exclusive choices, while [9] consider stochastic activity durations. None of these approaches considers resource unavailabilities or the emergence of new process instances.

A. Batch allocation

In contrast to offline approaches, online approaches assume that information becomes gradually available over time, e.g., new tasks become active during run-time. Many of these approaches propose batch allocation strategies, where an allocation decision is postponed until a number of tasks have become active [10–12]. E.g., [12] propose the k-batching allocation strategy, where an allocation decision is postponed until k tasks have become active. Once k tasks have become active, the expected processing times of all tasks across the resources are estimated. A schedule is then obtained by formulating a *parallel machines scheduling problem* to minimize the makespan, i.e., the total time required to finish all tasks. When large k values are chosen, scheduling decisions are based on more complete information, but on the other hand, choosing large k values can lead to situations that leave resources unnecessarily idle, e.g., when a resource has an empty worklist and fewer than k tasks have arrived such that no allocation is conducted.

B. Predictive allocation

In contrast to batch allocation approaches where only tasks that have already become active are considered, [13] presents an approach in which predicted future tasks are considered. In their approach, they assume that process instances come with different priorities. They formulate the allocation problem as an *assignment problem* in which they consider the available resources and the currently active and, in addition to that, predicted upcoming tasks. Tasks that belong to a high-priority instance incur a lower allocation cost. Thereby, predicted upcoming tasks, e.g., from a high-priority instance, can be allocated to a resource, leaving the resource effectively idle until the predicted task has become active. Their approach does not take resource unavailabilities into account: A task might still be allocated to a resource that will take overly long on it, even though it would be meaningful to wait with the allocation until a better resource has become available.

C. Reinforcement learning based approaches

Other works have proposed reinforcement learning techniques for learning resource allocation policies. [14] propose a Q-learning approach. As Q-learning suffers from state explosion due to the "course of dimensionality", [14] encode only a few parameters into the state space. Recently, some works have promoted the use of model-free reinforcement-learning techniques that overcome issues with the space-state explosion. [15] use Deep Reinforcement Learning (DRL) for training an allocation policy in the manufacturing domain. [16] present another DRL approach in which they add the possibility of non-allocating a task to the action space, thereby allowing to postpone an allocation. While DRL approaches seem to have the potential to grasp complex relationships in stochastic environments, they depend on a process simulator for training. Setting up such simulation models is cumbersome and remains a challenge in practice [17].

III. Online Resource Allocation Approaches

This section presents two online resource allocation approaches based on adaptations of the *assignment problem* (Sect. III-A) and the *parallel machines scheduling problem* (Sect. III-B). Both approaches are data-driven and only require a reasonable estimation of how long each resource will take for each task based on an event log. As a further assumption, this work considers only 1:1 allocations, i.e., a task can only be allocated to one resource, and a resource can only work on one task at a time. Furthermore, we do not allow for preemptions, i.e., a task's execution can not be paused and resumed later in time, or a task can not be reallocated to another resource once a resource has started processing it.

Definition 1 (Tasks and resources): Let T denote the set of active tasks, i.e., tasks that are ready to be allocated, and let $R := R_{available} \cup R_{unavailable} \cup R_{working}$ denote the set of resources. We distinguish available resources ($R_{available}$) that are idle, unavailable resources ($R_{unavailable}$) that are, e.g., out-of-office, and working resources ($R_{working}$) that are currently conducting a task. When a resource begins processing a task, it is removed from $R_{available}$ and added to $R_{working}$, and conversely, when it has finished the task, it is removed from $R_{working}$ and added to $R_{available}$. Similarly, when a resource becomes unavailable, it is removed from $R_{available}$ and added

to $R_{unavailable}$, and vice-versa when it becomes available again.

Definition 2 (Resource authorization): We define the set of resources authorized to conduct a task t as $R_{auth}(t) \subseteq R$.

Definition 3 (Expected (remaining) processing time): We define $c : T \times R \mapsto \mathbb{R}$ as the expected time a task $t \in T$ needs for completion when it is conducted by $r \in R_{auth}(t)$ and $c_r : R \mapsto \mathbb{R}$ as the time a resource $r \in R_{working}$ is yet expected to need for completing its currently allocated task.

A. Adapted assignment problem

The *assignment problem* can be defined as follows: Given a set of tasks T_A and a set of resources R_A, assigning a task $t \in T_A$ to a resource $r \in R_A$ incurs a cost w_{tr}. A task can only be allocated to one resource, and a resource can only be allocated to one task. The objective is to find an assignment with the maximum number of tasks that minimizes the total cost of the allocations. The problem can be solved in polynomial time, e.g., by formulating the problem as a minimum-cost maximum-flow problem of a bipartite graph. The goal of the proposed adaptation of the *assignment problem* is to allow the postponement of task allocation when no suitable resource is available. To achieve this, we introduce *dummy resources*. The idea is that a task can be assigned to a dummy resource when allocating it to a real resource instead is unsuitable, e.g., because it is expected to take all present resources disproportionately long to conduct the task. Hence, we define a set of dummy resources D where every task has its corresponding dummy resource. Moreover, we consider available and working resources for the allocations as working resources are expected to become available again after finishing their current tasks and the set of active tasks for T_A. Formally:

$$\begin{aligned} D &:= \{d_1, ..., d_{|T_A|}\} \\ R_A &:= R_{\text{available}} \cup R_{\text{working}} \cup D \\ T_A &:= T \end{aligned} \tag{1}$$

To solve the adapted *assignment problem*, we formulate it as a minimum-cost maximum-flow problem and create a flow network as follows: Let $G = (R_A \dot{\cup} T_A, E)$ be a bipartite graph where each resource is connected to all tasks it is authorized for via a directed edge and to one dummy resource. In Fig. 1 in the box denoted with a), we can see an example bipartite graph with $T_A = \{t_1, t_2\}$ and $R_A = \{r_A, r_B, d_1, d_2\}$. We assume that r_A is authorized to work on t_1 and t_2, while r_B is only authorized to work on t_1, resulting in the respective edges. For each of the two tasks, a dummy resource is added and connected to the task.

In order to form a minimum-cost maximum-flow problem, we i) add flow costs of an edge in the bipartite graph, i.e., the allocation cost w_{tr} and ii) logically add a source and a sink vertex, and connect the source to all resource vertices R_A with zero cost edges and all task vertices T_A to the sink vertex with zero cost edges. The capacity of all edges in the flow graph G' is 1. Solving the problem then yields the maximum number of tasks that can be allocated for a minimal cost.

Central to the adapted assignment problem is the allocation cost w_{tr} for $t \in T_A, r \in R_A$, which we define as:

$$w_{tr} = \begin{cases} c(t,r), \text{if } r \in (R_{auth}(t) \cap R_{\text{available}}) \\ c(t,r) + c_r(r), \text{if } r \in (R_{auth}(t) \cap R_{\text{working}}) \\ \delta \frac{1}{|R_{auth}(t)|} \sum_{r \in R_{auth}(t)} c(t,r), \text{if } r \in D \end{cases} \tag{2}$$

The cost of allocating a task to an available resource corresponds to the expected processing time $c(t,r)$ (cf. Def. 3). In contrast, the cost of allocating a task to working resources corresponds to the expected remaining processing time of its current allocation $c_r(r)$ plus the expected processing time $c(t,r)$. The cost of allocating a task to a dummy resource is defined as the factorized average processing time of all authorized resources on that task. The factor variable δ can be chosen as a hyperparameter and plays a central role as it impacts the likelihood that a task is allocated to its dummy resource instead of a real resource. Informally, it states how much faster a resource must expected to be than the average of all resources to be considered for allocation. $\delta = 0.5$, for example, means that a resource must be expected to perform a task at least twice as fast as the average of all authorized resources, while $\delta = 1$ means that a resource must be at least average. Theorem 1 formulates a lower bound for an allocation of a task to a resource based on δ.

Theorem 1 (Assignment problem dummy resource theorem): A resource $r \in R$ will never be allocated to a task $t \in T$ if

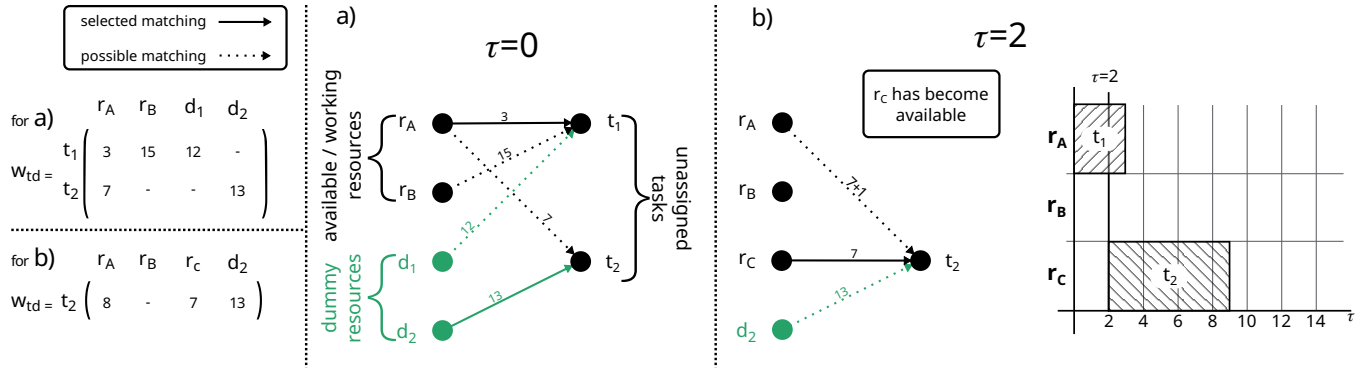


Fig. 1: Assignment problem with dummy resources

r is expected to take more than $\frac{1}{\delta}$ times longer to process t than the average of all authorized resources.

Proof 1: To proof Theorem 1 by contradiction, we assume that allocating any task $t \in T$ to a real resource $r \in R$ can be a solution in our minimum-cost maximum-flow problem. Since $|D| = |T|$, and every task vertex $t \in T$ is connected to exactly one dummy resource vertex $d \in D$, and $\forall d \in D(deg(d) = 1)$, and since every resource vertex is connected to the source vertex and every task vertex to the sink vertex and all edges have a capacity of one, a solution to the minimum-cost maximum-flow problem must include exactly one incoming edge on every task vertex. Therefore, t must either be allocated to a real resource $r \in R_A \backslash D$, or to its dummy resource $d \in D$. As $\forall d \in D(deg(d) = 1)$ holds, selecting a task's dummy resource does not affect the allocation cost of any other task or resource. As the objective is to minimize the flow cost, a resource r will only be selected when it incurs a lower cost than selecting the dummy resource. Hence, for allocating a resource r to a task t, $w_{tr} \leq w_{td}$ must hold.

For a resource r that is expected to take more than $\frac{1}{\delta}$ times longer on t than the average of all authorized resources $R_{auth}(t)$, $\frac{1}{\delta} w_{tr} > \frac{1}{|R_{auth}(t)|} \sum_{r \in R_{auth}(t)} c(t, r)$ holds. By plugging in w_{td}, we get $w_{tr} > w_{td}$, which contradicts $w_{tr} \leq w_{td}$.

From Theorem 1 follows that a task will never be allocated when only one authorized resource exists and $\delta < 1$ is chosen.

Fig. 1 shows an example of a resource allocation with our adaptation of the *assignment problem*. At time step $\tau = 0$ shown in a) the resources $R = \{r_A, r_B\}$ are available and the tasks $T = \{t_1, t_2\}$ need to be allocated. The solution to the *minimum-cost maximum-flow problem* yields an allocation of t_1 to r_A and of t_2 to its dummy resource d_2. At time step $\tau = 2$ the resource r_C becomes available. A new allocation is conducted. r_A and r_C are both expected to take 7 timesteps to process t_2, but since r_A is still working, its allocation cost is higher than that of r_C. Hence, t_2 is allocated to r_C.

B. Adapted parallel machines scheduling problem

In the *(unrelated) parallel machines scheduling problem*, a set of tasks T_P are allocated to a set of resources R_P to optimize a specific objective. A classical objective is minimizing the makespan, i.e., the total time required to finish all tasks, which makes the problem NP-hard [18]. However, Constraint Programming (CP) solvers often find near-optimal solutions within a short time [19]. In contrast to conducting allocations by solving the *assignment problem* (Sect. III-A), where only the costs of the most suitable tasks are considered, the *parallel machines scheduling problem* takes the costs from allocating all tasks to resources into account, i.e., it has a planning horizon.

Instead of explicitly adding dummy resources to the set of resources, we will now add a non-allocation option to the cost function. We define the set of resources and tasks as follows:

$$\begin{aligned} R_P &:= R_{\text{available}} \cup R_{\text{working}} \\ T_P &:= T \end{aligned} \tag{3}$$

The option to allocate tasks to dummy resources is implemented as a decision variable $Y = (y_1, ..., y_{|T_P|})$, i.e., $y_t = 1$ means that the task t will be allocated to a dummy resource. Alternatively, allocating tasks to real resources is denoted in the decision variable $X = (x_{1,1}, ..., x_{|T_P|,|R_P|})$. A CP formulation is given in Eq. 4:

$$\min \quad C = k_m + k_f + \sum_{t \in T_P} y_t c_d(t) \tag{4a}$$

$$\text{where} \quad c_d(t) = \delta \frac{1}{|R_{auth}(t)|} \sum_{r \in R_{auth}(t)} c(t, r) \tag{4b}$$

$$k_m = \max(\{c_r(r) + \sum_{t \in T_P} (x_{tr} c(t, r)) : \forall r \in R_P\}) \tag{4c}$$

$$k_f = \frac{1}{|R_P|} \sum_{r \in R_P} (k_m - (c_r(t) + \sum_{t \in T_P} (x_{tr} c(t, r)))) \tag{4d}$$

$$\text{subject to} \quad \sum_{r \in R_P} (x_{tr}) + y_t = 1 \qquad \forall t \in T_P \tag{4e}$$

$$x_{tr} = 0 \quad \forall r \in R_P \setminus R_{auth}(t), \forall t \in T_P \tag{4f}$$

$$x_{tr}, y_t \in \{0, 1\} \tag{4g}$$

The problem's objective is to minimize the cost function C and is subjected to the constraint in Eq. 4e, which ensures that every task is either allocated to a real resource R_P or its dummy resource, and the constraint Eq. 4f, which ensures that a task is only allocated to an authorized resource.

1) Cost functions: The cost function C consists of the primary objective of reducing the makespan k_m, a secondary fairness objective k_f that aims to balance the workload of the different resources, and the cost of allocating tasks to their dummy resources. We added the resource fairness objective, as long-taking tasks on a resource might dominate the makespan objective. Without the fairness objective this can lead to several equally optimal solutions from which some are undesirable: For example, solutions where some resources are left completely idle while others are allocated to multiple tasks.

In this adaptation, the δ variable no longer has the same intuitive meaning as in the previous adaptation: A resource that is expected to take more than $\frac{1}{\delta}$ times longer on a task t than the average of all authorized resources can still be allocated to this task, e.g., when such an allocation will otherwise not affect the makespan objective.

2) Task order: A solution to the problem's formulation can include multiple task allocations to a single resource. Therefore, one can select the order in which the tasks are processed. We implemented the *shortest processing time first* rule, i.e., every resource processes the task with the lowest expected processing time first. As this maximizes the frequency at which resources become available again, this strategy aims at creating

numerous new allocation opportunities. This appears beneficial in uncertain environments, where better schedules can emerge suddenly, e.g., when a new resource becomes available.

Fig. 2 shows the resource allocations obtained using the same setting as in Fig. 1. At time step $\tau = 0$, resources r_A and r_B are available, and t_1 and t_2 need to be allocated. Both tasks are (tentatively) allocated to r_A. t_1 is executed first on r_A as it has a lower expected processing time. At time step $\tau = 2$, resource r_C becomes available, which is expected to process t_2 equally fast as r_A. Similar to Fig. 1, allocating t_2 to r_C yields a lower cost than allocating t_2 to r_A.

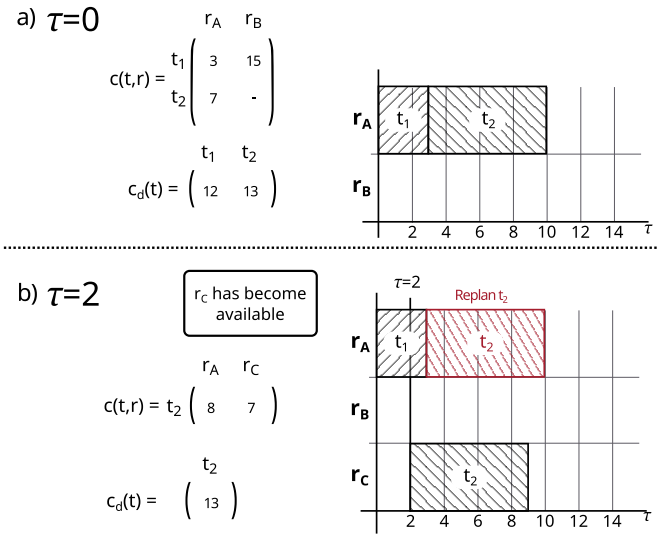


Fig. 2: Parallel machines scheduling problem's adaptation

IV. Implementation And Evaluation

We implemented the online allocation approaches presented in Sect. III, the approach by Park and Song [13], a k-batching approach, as well as the simple allocation mechanisms proposed in [2], namely Round-Robin allocation (R-RRA), Random allocation (R-RMA), and Shortest Queue First allocation (R-SHQ)[1]. We evaluated the approaches based on two process simulation models.

A. Implementation

This section will describe how we implemented the predictive models, the different allocation strategies, and the process simulation models.

1) Predictive models: For all approaches that require a task processing time prediction model, we used a fully connected neural network pre-trained on the task processing times obtained from an event consisting of three years of simulating the business processes. The approach from [13] additionally requires a next-task prediction, for which we used the same probabilistic model as in the process simulation.

2) Assignment problem's adaptation: We used the scipy[2] library to solve our adaptation of the *assignment problem*. The scipy library has implemented a modified variant of the Jonker-Volgenant algorithm, which has a worst-case time complexity of $\mathcal{O}(N^3)$ where $N = max(T_A, R_A)$ [20]. In benchmarks, the algorithm has been shown to yield results for large settings, e.g., $N = max(1000, 500)$ within milliseconds (c.f. [20]).

3) Parallel machines scheduling problem's adaption: We solved the CP formulation of our *parallel machines scheduling problem* using the CP solver from Google OR tools[3] (v9.8). Since the problem is NP-hard, finding optimal solutions is often not computationally feasible. However, the used CP solver was able to find most of the time near-optimal solutions within less than a second. Therefore, we limit the computation time for obtaining a solution to 2 seconds. In rare cases, the 2 seconds time limit was insufficient for finding solutions that are at least equally good as solutions obtained from a greedy scheduling approach. For these cases, we resorted to a backup solution, namely finding an allocation obtained by applying our adaptation of the *assignment problem*.

4) K-Batching [12]: For comparison to our approaches, we implemented k-batching. Since the approach from [12] does not consider resource unavailabilities, we implemented the approach as follows: An allocation is postponed until k tasks have arrived. Then, the task processing times are obtained across all resources, and a *parallel machines scheduling* problem is solved for which the k tasks and all available and working resources are considered. Every task assigned to a resource is then added to a resources worklist. When a resource becomes unavailable, all tasks on its worklist are transferred back to the set of unassigned tasks.

5) Park & Song's approach [13]: We also implemented [13] by adding the most likely next task for every process instance to the set of allocatable tasks. The cost for these expected next tasks was set, in line with [13], as the expected remaining time of their preceding tasks plus their expected execution times on the respective resource. All process instances were given the same priority in our implementation.

6) Process simulation: The implemented approaches are compared based on process simulation. We created two simulation models, which were (partly) discovered from event logs. For both simulation models, the control-flow perspective was discovered as a frequency directly follows graph. The instance arrival times were simulated from gamma distributions fitted on the respective event logs.

The first event log used for creating a process simulator was the synthetic *purchasing-example (PE-X)* event log, which is part of academic material provided by Fluxicon[4]. To demonstrate our approach on this dataset, we manually specified some properties of this simulation model: First, since the total number of resources conducting tasks in a timeframe, e.g. in each hour of a day, has shown to be near constant in

[1]Prototype in Python available at https://github.com/ltsstar/BPM_Resource_Allocation

[2]https://scipy.org/

[3]https://github.com/google/or-tools

[4]https://fluxicon.com/academic/material/

this dataset, we added a synthetic resource availability profile that is supposed to resemble real-world resource availabilities: Resources are only available during working days, at which the number of available resources grows in the early morning hours and starts declining in the afternoon. Second, the processing times of tasks have also shown to be similar across resources. Hence, we defined task-resource processing times to randomly follow one from five normal distributions with means between 30 min and 6 hours. The second event log used was the *BPIC-2017*[5] real-world event log from a loan application process in a Dutch financial institute. A recurring biweekly resource availability pattern was obtained from the event log and used in the process simulation. The processing times were estimated from a pre-trained fully connected neural network. To account for variability in the simulation, we added normally distributed random errors to the processing times.

B. Evaluation

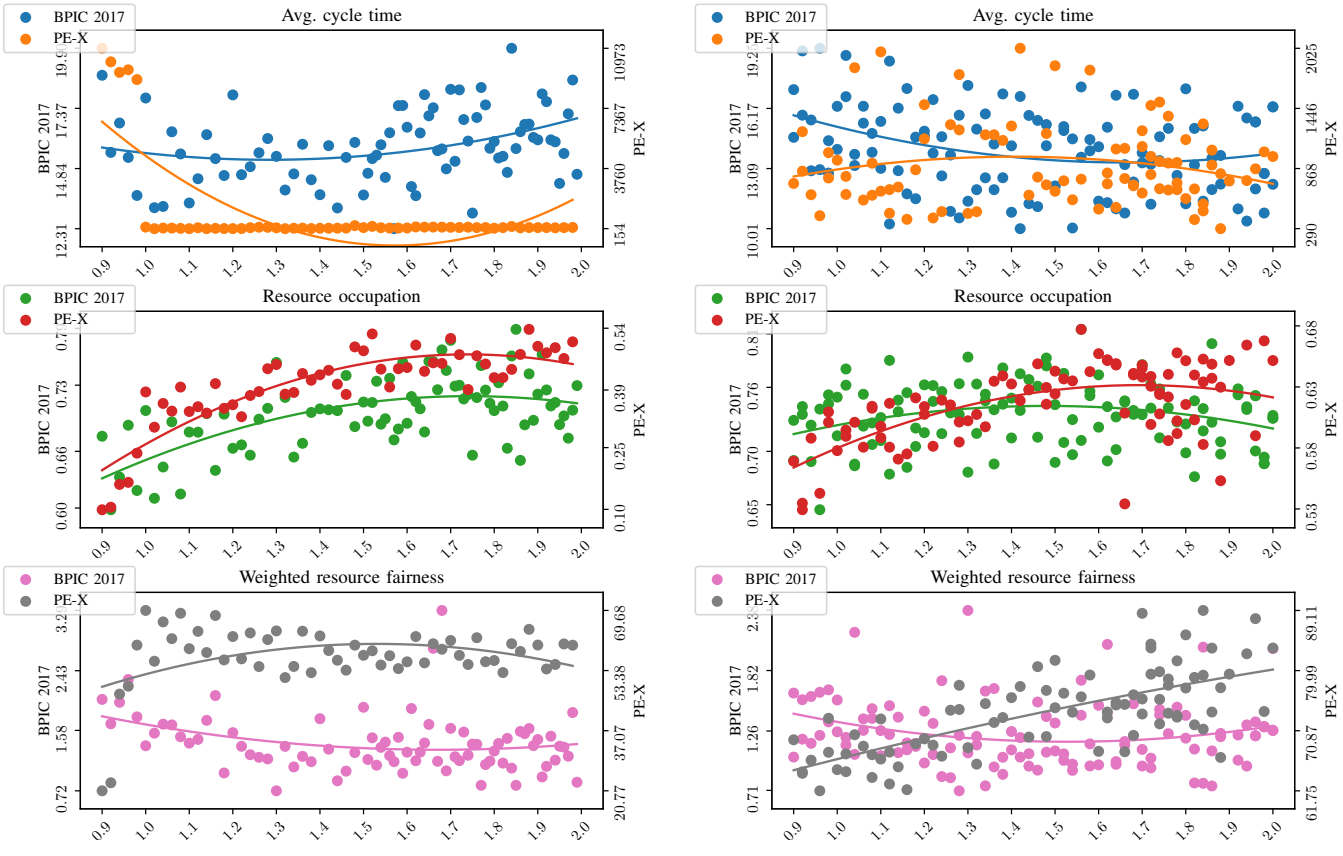
We compared the implemented approaches on three metrics, namely the *average cycle time*, the mean time from a process instance's arrival until its completion, which is a typical performance metric (c.f. [3]), and two resources related metrics: The *average resource occupation*, defined as the mean proportion of time that resources are working during their presence, and the *weighted resource fairness*, calculated as the total deviation from the mean resource occupation, weighted by the resources' actual presence (c.f. [21]). In this work, we consider low values in all metrics as desirable.

1) Impact of the δ parameter: We first evaluated whether a chosen δ value impacts the metrics in our adaptations. Based on the intuitive meaning of the δ parameter in the *assignment problem's* adaptation, we conducted simulations with δ-values in the range $(0.9, 2)$ for both adaptations. Fig. 3 shows the effects of the δ parameter on the three evaluated metrics on both simulation models. The results show that the chosen δ parameter impacts the evaluated metrics in both adaptations and simulation models.

In the *assignment problem's* adaptation, the *PE-X* simulation shows a sudden change in all metrics when the δ value exceeds 1. This sudden change might be explained due to the simpler properties of the *PE-X* simulation model: First, the task durations follow only one of five normal distributions, and second, the *PE-X* simulation model is only concerned with 27 resources in total. In contrast, the processing times in the *BPIC-2017* simulation model are obtained from a more complex model and the simulation consists of more resources (114).

As higher δ-values increase the cost of allocating a task to a dummy resource, and thereby, increase the likelihood that

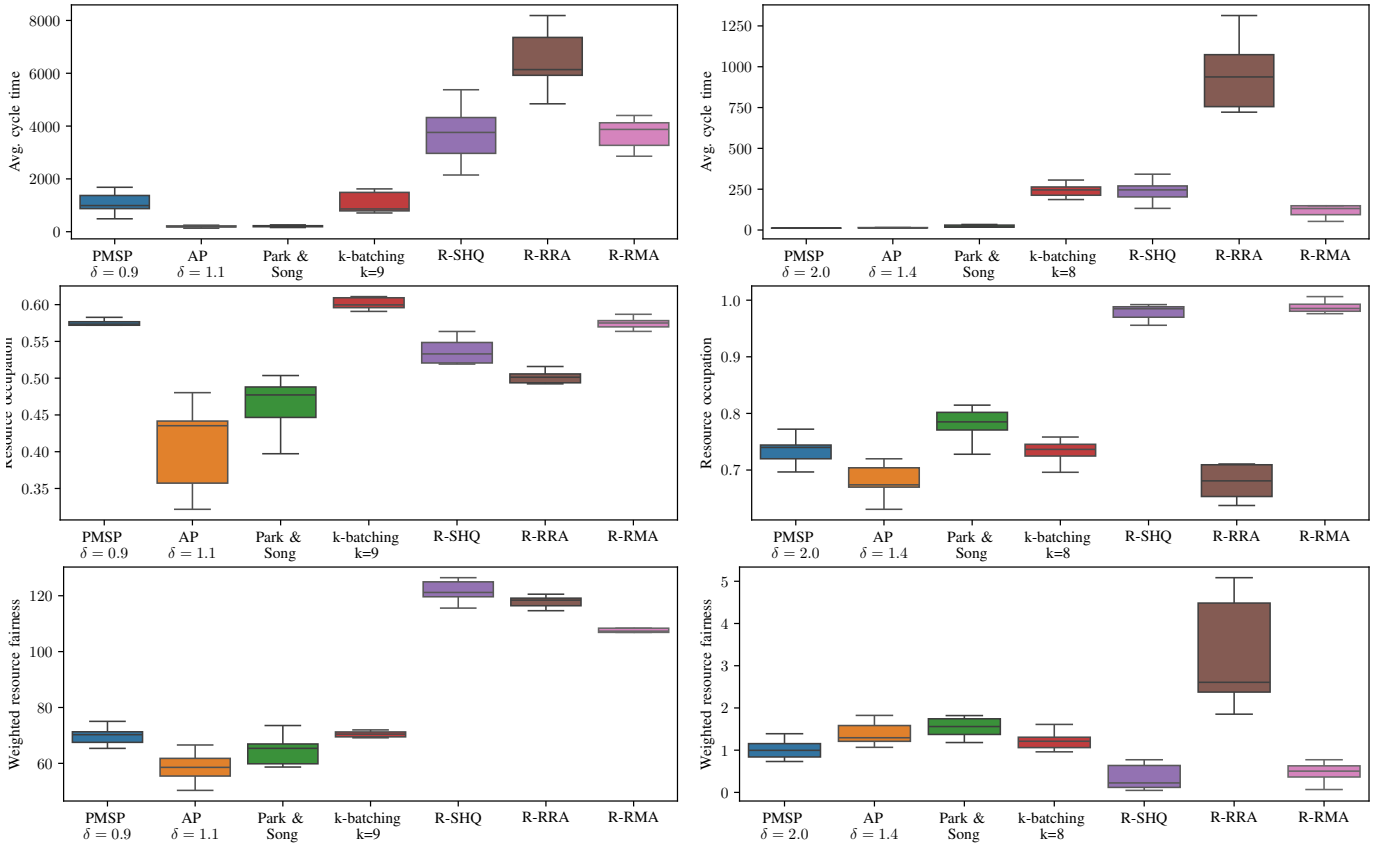
[5]https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f



(a) Assignment problem's adaptation

(b) Parallel machines scheduling problem's adaptation

Fig. 3: Impact of the δ parameter



(a) PE-X simulation

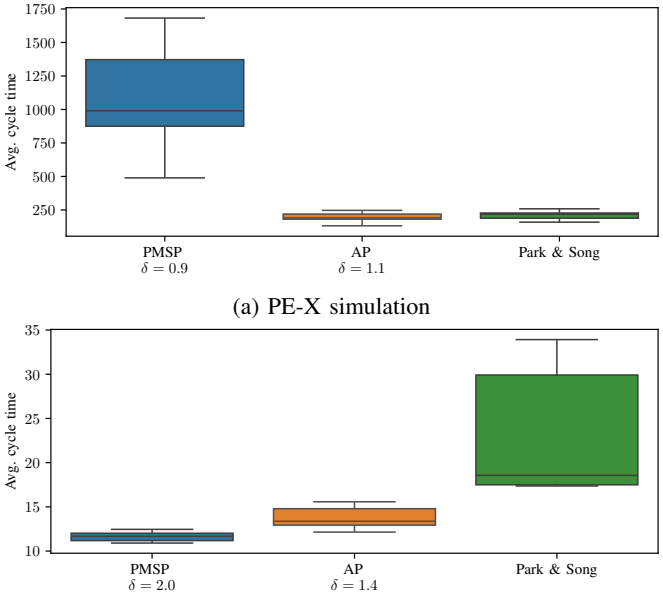
(b) BPIC-2017 simulation

Fig. 4: Comparison of different approaches (PMSP: *parallel machines scheduling problem's* adaptation, AP: *assignment problem's* adaptation)

a task is allocated to a less adequate resource, we expected larger δ-values to positively correlate with the *average cycle time* and *average resource occupation* metrics. We can see such a correlation for the *task assignment problem's* adaptation and the *BPIC-2017* simulation. No correlation can be seen for both metrics in the *PE-X* simulation and the *parallel machines scheduling problem's* adaptation. The extent of evaluated δ-values might not have been sufficient to capture a correlation in the *parallel machines scheduling problem's* adaptation and for the *PE-X* dataset.

2) Comparison to other approaches: Based on the findings from Fig. 3, we selected for the *BPIC-2017* simulation $\delta = 1.4$ for the *assignment problems's* adaptation and $\delta = 2.0$ for the *parallel machines scheduling problem's* adaptation. For the *PE-X* simulation, we selected $\delta = 1.1$ for the *assignment problem's* adaptation and $\delta = 0.9$ for the *parallel machines scheduling problem's* adaptation for further comparison to other approaches. Similarly, we selected the best k values with respect to the *average cycle time* metric for the k-batching approach with $k = 9$ for the *PE-X* simulation and $k = 8$ for the *BPIC-2017* simulation, respectively. We ran 10 simulations for every resource allocation approach. The results are shown in Fig. 4.

The results show that the approach from [13] and our



(a) PE-X simulation

(b) BPIC-2017 simulation

Fig. 5: Comparison between our approaches and [13]

approaches achieve the best *average cycle times*, followed by k-batching and the simple allocation approaches. For better readability, Fig. 5 provides a comparison of the *average cycle time* metric between our approaches and [13]. In the *PE-X* simulation, our adaptation of the *assignment problem* incurs overall the lowest *average cycle time*, *resource occupation*, and *weighted resource fairness* values, followed by [13]. In the *BPIC-2017* simulation, the adaptation of the *parallel machines scheduling* problem incurs the best *average cycle times*, but at a higher *resource occupation* than [13].

Compared to [13], our approaches perform better on the *BPIC-2017* simulation. That might be due to a higher control-flow uncertainty in the *BPIC-2017* simulation model: While activities in the *PE-X* process model often occur sequentially, the *BPIC-2017* simulation model possesses more randomness in the order of activities.

As [13] bases allocation decisions on the next tasks, a higher control-flow uncertainty will likely reduce the allocation qualities from their approach. This indicates that our approach is especially well-suited for scenarios with a high control-flow uncertainty.

V. Conclusion

In this work, we presented adaptations of the *assignment* and *parallel machines scheduling problem* to address the problem of uncertainty due to (un)availability of resources during runtime. Based on the introduction of dummy resources, our approaches can postpone allocations when no suitable resource for processing a task is available. Both approaches can directly be applied in PAISs without the need for previous training on complicated simulation models as long as the tasks' processing times can be estimated. Our evaluations on two business process simulation models, which aim to resemble realistic business processes, show that our approaches can outperform other resource allocation approaches. Our approaches seem to be especially well-suited when upcoming tasks are hard to predict. Our evaluation shows that the δ hyperparameter impacts performance metrics. Due to the computational expenses that come with simulation, we scanned only a range of hyperparameter values. Especially for our adaptation of the *parallel machines scheduling problem*, it remains unclear whether even better results could have been achieved by choosing other values. Future work will consider allocations of multiple resources to one task and resources working on multiple tasks. Moreover, we plan to apply search techniques, e.g. simulated annealing, for hyperparameter optimization. Furthermore, we plan to exploit control-flow information and evaluate the approaches in real-world scenarios.

References

[1] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management, Second Edition*. Springer, 2018.

[2] N. Russell, W. M. P. van der Aalst, A. H. M. ter Hofstede, and D. Edmond, "Workflow resource patterns: Identification, representation and tool support," in *Advanced Information Systems Engineering*, 2005, pp. 216–232.

[3] W. M. P. van der Aalst, *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.

[4] S. Rostami, S. Creemers, and R. Leus, "New strategies for stochastic resource-constrained project scheduling," *J. Sched.*, vol. 21, no. 3, pp. 349–365, 2018.

[5] F. Ballestín and R. Blanco, "Theoretical and practical fundamentals for multi-objective optimisation in resource-constrained project scheduling problems," *Comput. Oper. Res.*, vol. 38, no. 1, pp. 51–62, 2011.

[6] L. Pufahl, S. Ihde, F. Stiehle, M. Weske, and I. Weber, "Automatic Resource Allocation in Business Processes: A Systematic Literature Survey," Mar. 2024. [Online]. Available: http://arxiv.org/abs/2107.07264

[7] G. Havur, C. Cabanillas, and A. Polleres, "Benchmarking answer set programming systems for resource allocation in business processes," *Expert Syst. Appl.*, vol. 205, p. 117599, 2022.

[8] W. M. P. van der Aalst, "Petri net based scheduling," *OR Spektrum*, vol. 18, no. 4, pp. 219–229, Dec. 1996.

[9] K. F. Doerner, W. J. Gutjahr, G. Kotsis, M. Polaschek, and C. Strauss, "Enriched workflow modelling and stochastic branch-and-bound," *Eur. J. Oper. Res.*, vol. 175, no. 3, pp. 1798–1817, 2006.

[10] P. Delias, A. D. Doulamis, N. D. Doulamis, and N. F. Matsatsinis, "Optimizing resource conflicts in workflow management systems," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 3, pp. 417–432, 2011.

[11] M. Arias, J. Munoz-Gama, M. Sepúlveda, and J. Miranda, "Human resource allocation or recommendation based on multifactor criteria in on-demand and batch scenarios," *European Journal of Industrial Engineering*, vol. 12, no. 3, pp. 364–404, 2018.

[12] D. D. Zeng and J. L. Zhao, "Effective role resolution in workflow management," *INFORMS J. Comput.*, vol. 17, no. 3, pp. 374–387, 2005.

[13] G. Park and M. Song, "Prediction-based resource allocation using LSTM and minimum cost and maximum flow algorithm," in *Process Mining*, 2019, pp. 121–128.

[14] Z. Huang, W. M. P. van der Aalst, X. Lu, and H. Duan, "Reinforcement learning based resource allocation in business process management," *Data Knowl. Eng.*, vol. 70, no. 1, pp. 127–145, 2011.

[15] K. Zbikowski, M. Ostapowicz, and P. Gawrysiak, "Deep reinforcement learning for resource allocation in business processes," in *Process Mining Workshops*. Springer, 2022, pp. 177–189.

[16] J. Middelhuis, R. L. Bianco, E. Scherzer, Z. A. Bukhsh, I. J. B. F. Adan, and R. M. Dijkman, "Learning policies for resource allocation in business processes," *CoRR*, vol. abs/2304.09970, 2023.

[17] I. Beerepoot and et al., "The biggest business process management problems to solve before we die," *Comput. Ind.*, vol. 146, p. 103837, 2023.

[18] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Springer International Publishing, 2022.

[19] Y. Bukchin and T. Raviv, "Constraint programming for solving various assembly line balancing problems," *Omega*, vol. 78, pp. 57–68, 2018.

[20] D. F. Crouse, "On implementing 2d rectangular assignment algorithms," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 52, no. 4, pp. 1679–1696, 2016.

[21] M. Arias, J. Munoz-Gama, and M. Sepúlveda, "Towards a taxonomy of human resource allocation criteria," in *BPM Workshops*, vol. 308. Springer, 2017, pp. 475–483.