Instance Configuration and Scheduling Based on the Resource-Augmented Process Structure Tree

Felix Schumann(✉), G. Wessel van der Heijden, and Stefanie Rinderle-Ma

Technical University of Munich, TUM School of Computation, Information, and Technology, Garching, Germany
{felix.schumann,wessel.heijden,stefanie.rinderle-ma}@tum.de

Abstract. Real-world applications often require the resource-specific configuration of process instances based on system states. In clinical processes, for example, the number and order of documentation tasks depend on whether they are conducted by an intern or the head physician. One approach supporting resource-specific instance configuration is the Resource-Augmented Process Structure Tree (RA-PST). It also offers the flexibility to drive the configuration through optimized resource allocation. In particular, combinatorial optimization approaches can be explored to plan optimal combinations of multiple configurable process instances into a system. We use Mixed Integer Programming (MIP) to find optimal process configurations and Constraint Programming (CP) to schedule the process instances. To combine configuration and scheduling of process instances, we propose an integrated CP formulation as well as a Logic-Based Benders Decomposition (LBBD) inspired by the integrated process planning and scheduling problem (IPPS). The approaches are applied and evaluated in an offline and an online scheduling setting. We show the suitability of the CP and LBBD formulation for the configuration and scheduling problem. The integrated CP can find optimal solutions for small problems but struggles to prove optimality in a timely manner when the problem size grows. Here, the LBBD can identify better lower bounds. For the online setting, the integrated CP finds optimal solutions for most process instances while considering the planned resource availabilities in the system.

Keywords: Process Configuration · Resource Allocation · Scheduling · Constraint Programming · Logic-Based Benders Cuts

1 Introduction

Process-Aware Information Systems (PAIS) and the underlying real-world processes in applications such as manufacturing or clinical services move towards flexible processes [10,14]. Configurable process models enable process modelers to incorporate flexibility at design time and allow them to better reflect reality.

A. Senderovich et al. (Eds.): BPM 2025, LNBIP 564, pp. 110–127, 2026.
https://doi.org/10.1007/978-3-032-02929-4_7

When executing the process models, multiple process instances are in competition over the human and non-human resources needed to execute their tasks. Configurable process models, such as the RA-PST formalized in [17], can be utilized to bypass resource bottlenecks by adapting their control-flow configuration to the current resource availability. Approaches from combinatorial optimization can leverage this flexibility to optimize process performance and resource utilization in a system.

The RA-PST intertwines the flexibility required by real-world processes with a representation of possible resource allocations for each task and ensures the structural soundness of the resulting process models. At design time, the RA-PST allows for separate modeling of the "core" Process Structure Tree (PST), which represents the business logic and the separate modeling of resources that later execute the tasks (resource augmentation). Based on the PST, an RA-PST is executable by a process engine [17,21]. So far, no approach addresses the integrated configuration and scheduling of process instances to optimize process performance. Hence, in this paper, we propose and compare three configuration and scheduling approaches shown in Fig. 1 based on the RA-PST to find a combination of process configurations that optimizes an objective function. While approach a) separates configuration and scheduling for each instance, approaches b) and c) utilize the provided flexibility by configuring and scheduling multiple process instances simultaneously. The proposed mathematical programming models are inspired by optimal solution approaches for the Integrated Process-Planning and Scheduling Problem (IPPS) from Operations Research (OR). The proposed optimization models enable the assessment of the quality of a found solution w.r.t. optimality by specifying an optimality gap through an upper and a lower bound.

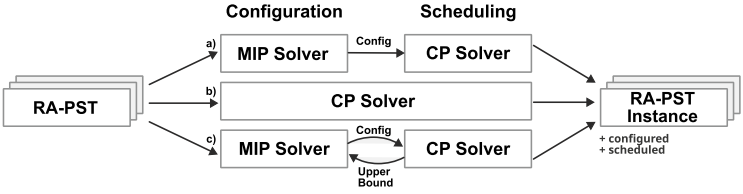

Fig. 1. Different solution approaches for the combined configuration and scheduling problem with RA-PSTs. a) separated approach, b) integrated CP approach, c) Logic-Based Benders Decomposition.

In Fig. 1b) we employ Constraint Programming (CP) to solve an integrated version of the problem, which configures and schedules the instances simultaneously. CP has grown in attention in recent years to solve scheduling problems in Operations Research (OR) [11] and Business Process Management (BPM) [18]. It performs specifically well on sequencing problems. In pure assignment decisions, Mixed Integer Programming (MIP) models perform better [11]. The

approach in Fig. 1c) is based on the IPPS, e.g. [1,10]. By separating configuration and scheduling, the strengths of both approaches can be fostered for the problem at hand. We present a novel Logic-Based Benders Decomposition (LBBD) of the problem, where the configuration is solved through a MIP, and the scheduling through a CP. In the LBBD, information is exchanged between both optimization steps. Solving both problems in a sequential manner as in [9] is presented in Fig. 1a). As described in [11], the integration of configuration and scheduling is an open challenge and an active field of research for optimal solution approaches. In coherence with this research field, we work with purely sequential processes. We propose three solution approaches for the integrated configuration and scheduling problem. The proposed approaches can find optimal solutions to the problem and report an optimality gap for any found solution. By basing the approach on the RA-PST process model, we generate configured, executable process instances with allocated resources. Each process instance is configured in context with all other process instances to optimize makespan.

The remainder of this paper is structured as follows: Sect. 2 defines preliminaries for RA-PSTs. Section 3 analyzes the complexity of the configuration problem, and formalizes the optimization methods. These methods are applied in an offline and an online setting in Sect. 4. Section 5 discusses the findings in the context of related work, and 6 concludes the paper.

2 Preliminaries and Example

To combine a structural process model with resource allocation, we utilize the *RA-PST* [17] that augments a process structure tree [21] with the flexible allocation of resources. Figure 2a) shows an example taken from the diagnostic process of a radiology department. The PST reflects a sequence of two tasks, i.e., `report` and `approve` where `report` can be processed by a resource with roles L_1, L_2, L_3. Let R be the set of resources (cf. Figure 2b). The RA-PST creates flexibility by enabling resources to apply changes to the control flow in case they are allocated to a task, e.g., inserting an additional `read` task if the allocated resource is an intern. To this end, a resource $r \in R$ can have one or several *resource profiles* $rp := (r, role, t, Attr, cp)$ where r has role $role$ to specify access rights and is assigned to task t in the PST. r can be equipped with attributes $Attr$ and change patterns $cp \in \{\text{insert, delete, replace}\}$, which modify the PST if r is allocated to t. In the example, resource `p` has two resource profiles rp_2 and rp_3. `i` has resource profile rp_1 referring to task `report` and specifying an insert of task `read` after `report`. $Attr$ typically contains the costs for allocating a resource.

The RA-PST is constructed by appending branches to each leaf task t in the PST (cf. Figure 2c). A branch consists of the resource node r allocated to t, the corresponding resource profile node, and potentially the associated change pattern node referring to a task t'. If the change pattern inserts t', the branch might be expanded. For the example, three branches are appended to task `report`, one for each resource `i, p, h` having the associated roles `L1, L2, L3`. Take the branch for `i`: below node `i`, the associated resource profile rp_1 is appended and

below that, the change pattern node for insertion of `read`. For `read`, resource `p` is authorized and appended, followed by the resource profile rp_3. Here, the branch ends as rp_3 does not specify a change pattern.

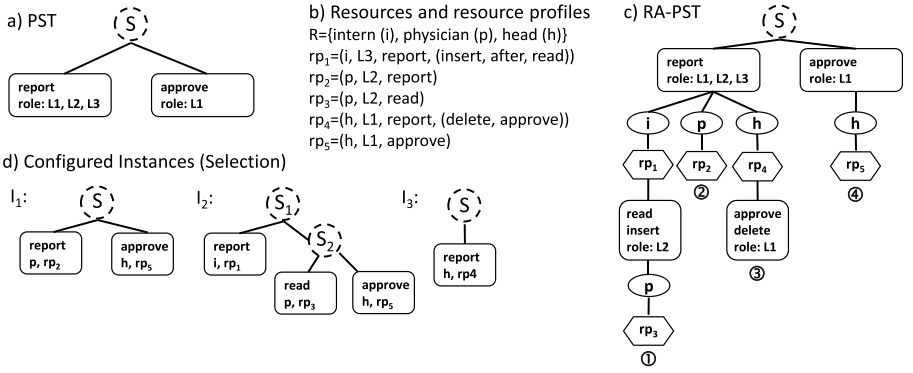

Fig. 2. RA-PST representation of the diagnostics process in a radiology department. a) Process as PST, b) Resource description, c) full RA-PST with branches, d) Possible configured process instances from the RA-PST.

As shown in [17], a branch is *valid*, i.e., can be employed for *process instance configuration*, if it ends in a resource profile or a delete pattern. A process instance configuration is achieved by choosing a branch assignment for each PST task and then resolving all change patterns in the selected branches. Figure 2d) shows three possible process instance configurations I_1, I_2, and I_3 for the RA-PST in Fig. 2c). For I_2, as `i` is allocated to `report`, task `read` is inserted after `report`. For I_3, as `h` is allocated to `report`, task `approve` is deleted. These configurations are sound due to the PST soundness, and the resource allocation is possible due to the validity property. For the example, several configurations are possible. In the real-world application behind this example, a traditional process model containing all variants would become complex. The RA-PST declutters the model and pushes the flexibility to the resources which are the root cause for variants. Hence, a separation of process perspectives is achieved.

The choice of branches can depend on different aspects, such as costs modeled as attributes in the resource profiles, and can be solved as an optimization problem. So far, the scope of the allocation in [17] has been limited to single process instances. In the following, we will tackle the combined configuration and scheduling of multiple process instances $\mathcal{I}$ as an optimization problem to decide on the optimal resource allocation among all tasks and process instances.

For the remainder of this paper, the term "instance" is used in its BPM context, where it denotes an instance of a process. We refer to each resource allocated to a task as a *job* $j \in J$ that has an individual cost c_j where J is the set of all jobs. A branch $b \in B$ represents each unique path from the root node to a leaf, where B is the set of all branches. Each branch contains a set

of jobs $J_b \subseteq J$. The overall cost c_b of a branch $b \in B$ can be calculated as the sum of all allocated resources in this branch, i.e. $c_b = \sum_{j \in J_b} c_j$. From here on, the costs represent the processing time of a job and for each job, this cost is predetermined.

For the RA-PST we rely on the validity notion given in [17], which only allows for the deletion of top-level tasks, and exclude invalid branches in the optimization problem. To ensure a sequential process model as in the IPPS problem [1], we limit the change patterns to deletion and insertion of tasks and do not allow for parallel or exclusive choices.

3 Configuration and Scheduling of RA-PST Instances

The optimization problem consists of finding a valid configuration for each process instance (configuration problem) and fitting each process instance into a system schedule (scheduling). We leverage the variability of RA-PSTs to improve the makespan of the system's schedule through combination of configurations. As an input, we are given a set of process instances $\mathcal{I}$. Each process instance $I \in \mathcal{I}$ contains a set of tasks $\mathcal{T}(I)$, a set of jobs $J(I)$, and a set of branches $B(I)$.

3.1 Configuration Problem

For the configuration problem alone, we consider a single instance I, and show that configuring this single instance alone is NP-hard. A branch can delete another task in the same process instance if it contains a delete change pattern for that task. In the configuration problem, for each task that is not deleted exactly one branch must be selected. If e.g., branch ③ is selected for `report` in Fig. 2, ①, ② and ④ cannot be selected. Definition 1 formally defines the minimum cost RA-PST configuration problem. Figure 3 gives an illustrative example of the configuration problem.

Definition 1 (Minimum cost RA-PST configuration problem). *Given are n branches $B = \{b_1, \ldots, b_n\}$ partitioned over k tasks $\mathcal{T} = \{T_1, \ldots T_k\}$, i.e. $T_i \subseteq B$. Each branch $b \in B$ has a cost c_b and a set of tasks $\tau(b) \subseteq \mathcal{T}$ deleted by branch b. Find the minimum cost set of branches $X \subseteq B$ such that for each task $T_i \in \mathcal{T}$ either*

1. *there is exactly one branch $b \in X$ such that $b \in T_i$*
2. *there is a $b \in X$ such that $T_i \in \tau(b)$ and for all $b' \in T_i$ it holds that $b' \notin X$.*

NP-Hardness. We reduce from the minimum weight maximal independent set problem (WMMIS), which was shown to be NP-hard by Demange [3].

Definition 2 (Minimum weight maximal independent set problem (WMMIS), [3]). *Given a graph $G = (V, E)$, an independent set is a subset $V' \subseteq V$ such that no two vertices in V' are linked by an edge E; an independent set V' is maximal if for every vertex $v \in V \backslash V'$, there exists $v' \in V'$ such that $(v, v') \in E$. Let w_v be the weight of vertex v for every vertex $v \in V$. For a set of vertices $V' \subseteq V$, its weight $w(V')$ is the sum of weights of the vertices in V'. Find a maximal independent set $V' \subseteq V$ with minimum weight $w(V')$.*

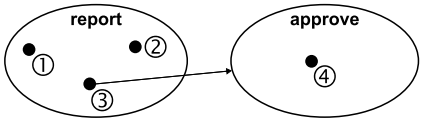

Fig. 3. A graph representation of an RA-PST instance of the example. Elements represent branches, sets represent PST tasks (Fig. 2a)), arrows indicate a delete change pattern. If branch ③ is chosen, no element from the set "approve" may be chosen.

Lemma 1. *If a polynomial time algorithm exists for Minimum Cost RA-PST configuration problem, then the minimum weight maximal independent set can be polynomially decided.*

Proof. Given is some WMMIS instance $G = (V, E)$. Index each vertex in V such that $V = \{v_1, \ldots, v_n\}$. Define branch b_i with $c_i = w_{v_i}$ for every vertex $v_i \in V$. Let task T_i be the set of branches only containing b_i for every $v_i \in V$. For every edge $(v_i, v_j) \in E$, add task T_j to $\tau(b_i)$. In the minimum cost RA-PST configuration problem, for any branch b_i that is selected, all tasks $\tau(b_i)$ are deleted. This means no branch that are part of any task in $\tau(b_i)$ can be selected. Therefore, for any b_i that is selected, no neighboring b_j with $(v_i, v_j) \in E$ can be selected, since T_j was deleted. This means that any solution $X \subseteq B$ to the RA-PST configuration problem is an independent set. By the definition of the minimum cost RA-PST, there must be a branch in each task T_i that is in X, or T_i is deleted. Therefore, X is a maximal independent set. Since the minimum cost RA-PST problem minimizes the cost of all selected branches, X is a minimum weight maximal independent set.

Demange showed that the WMMIS problem can not be approximated independently of the node weights [3]. This gives the following corollary for the minimum cost RA-PST configuration problem.

Corollary 1. *The minimum cost of the RA-PST configuration problem can not be approximated in polynomial time independently of branch weights, unless P=NP.*

We define the configuration problem as an ILP. Define the closed neighborhood of b as $N[b] = \bigcup_{T_i \in \tau(b) \cup \{T_j : b \in T_j\}} T_i$, which is the set of branches that can not be selected if branch b is selected, including branch b. The open neighborhood is defined as $N(b) = N[b] \backslash \{b\}$. If branch $b \in X$, then $b' \notin X$ for any $b' \in N(b)$. Since we have to select a maximal independent set, exactly one branch $b' \in N[b]$ must be selected. This gives the following ILP formulation of the RA-PST configuration problem.

$$\min \sum_{b \in B} c_b x_b \qquad \text{(Config)}$$

$$\text{s.t.} \sum_{b' \in N[b]} x_{b'} = 1 \qquad \forall b \in B \tag{1}$$

$$x_b \in \{0,1\} \qquad \forall b \in B$$

In this ILP, the variable x_b indicates whether branch b is selected, c_b is the cost of branch b, and constraint (1) ensures that exactly one branch of the neighborhood is selected.

3.2 Scheduling Problem

In the scheduling problem, a set of process instances $\mathcal{I}$ is given with selected branches X. All jobs $j \in J_b$ of each branch $b \in X$ must be scheduled on resource $r_j \in R$ adhering to the precedence constraints of each process instance. Any pair of jobs $i, j \in J(I)$ for all $I \in \mathcal{I}$ have a sorted order, which gives precedence constraints $i \prec j$. Furthermore, each resource $r \in R$ may process at most one job at any time. Each job j must be processed non-preemptively for processing length c_j. The objective is to find a schedule that schedules all selected jobs that adheres to the precedence constraints while minimizing the total makespan. The scheduling problem is a generalized job-shop scheduling problem, which is known to be NP-hard ([4], problem SS18). In the job-shop scheduling problem all individual jobs in a process instance must be scheduled on different resources [13], which is not a requirement for our scheduling problem. Since our scheduling problem is more general than job-shop scheduling, our scheduling problem is also NP-hard.

Consider the example in Fig. 2 for which two process instances i and j arrive simultaneously. Both instances are configured as variant I_3 (Fig. 2 d)). This configuration results in two tasks `report` that must be executed by resource `h`. Since one resource can only process one task at a time, resource `h` first processes task `report` for i before processing `report` for j. If a second resource `h` was available, both `report` tasks could be processed in parallel.

3.3 Integrated Constraint Programming Formulation

We combine the configuration and scheduling problem in a single formulation. The CP formulation is given in (ICP). The configuration constraint is formulated similarly as in the configuration problem in (Config). In the integrated version, the goal is to minimize the makespan instead of the sum of the branch costs.

Consider the goal of processing two process instances in the example over the given resources with minimum makespan. Configuring one process instance to I_1 and the other to I_3 can lead to a shorter overall makespan if the approval by `h` takes less time than writing a full `report`.

$$\min \max_{j \in \cup_{I \in \mathcal{I}} J(I)} (\text{EndOf}(Task_j)) \tag{ICP}$$

$$\text{s.t.} \quad Task_j = \text{IntervalVar}(c_j, optional) \qquad \forall I \in \mathcal{I},\ j \in J(I) \quad (2)$$

$$\text{NoOverlap}(Task_i, Task_j) \qquad \forall r \in R,\ i, j \in J_r \quad (3)$$

$$\text{EndBeforeStart}(Task_i, Task_j) \qquad \forall I \in \mathcal{I}, i, j \in J(I),\ i \prec j \quad (4)$$

$$\text{PresenceOf}(Task_i) = \text{PresenceOf}(Task_j) \qquad \forall I \in \mathcal{I}, b \in B(I),\ i, j \in J_b \quad (5)$$

$$\sum_{j_1 \in b' : b' \in N[b]} \text{PresenceOf}(Task_{j_1}) = 1 \qquad \forall I \in \mathcal{I},\ b \in B(I) \quad (6)$$

The objective of (ICP) is to minimize the makespan. The function "EndOf(x)" returns the end of interval variable x. Equation (2) defines the interval variables for each job in all process instances. "IntervalVar(a, *optional*)" returns an interval variable of length a for which it is optional whether it is present. Equation (3) ensures that each resource processes at most one task at any time. The constraint "NoOverlap(x, y)" ensures that no overlap between interval variables x, y exist. Equation (4) enforces the precedence constraints of the jobs. The constraint "EndBeforeStart(x, y)" ensures that interval variable x must have its end before the start of interval variable y. Equation (5) ensures that all jobs in the same branch are scheduled when one job in the same branch is selected. The boolean function "PresenceOf(x)" returns a 0 if interval variable x is not present, and a 1 if it is. Equation (6) is the configuration constraint, where the first job j_1 in a branch $b' \in N[b]$ has to be present. In combination with (5), this ensures exactly one branch is selected.

3.4 Logic Based Benders Decomposition

CP is especially effective for the scheduling part of the problem. It can, however, struggle with finding good configurations for the process instances, which is a strength of MIP formulations [11]. In order to leverage the strength of MIP formulations for the configuration problem and CP for the scheduling problem, we use LBBD, which is the state-of-the-art approach to find solutions for IPPS [1, 10]. We adapt the approach presented in [1] for our problem with the expectation to find better lower bounds than the (ICP). To this end, we decompose the problem into a master problem and a subproblem. The master problem solves the configuration problem. Additional constraints to the master problem provide a lower bound on the makespan for the selected combination of configurations. The jobs that are selected by the master problem are then scheduled by the CP. The CP returns a makespan for the specific configuration combination, for which benders cuts are added to the master problem. If the makespan is not equal to the lower bound, the master problem selects a new configuration combination with the minimum lower bound until the termination requirement has been reached.

The objective for the combined problem is to minimize the makespan $c_{\max}$. We pick a lower bound on the makespan $c_{\max}$ in order to aid picking the master problem when selecting an assignment. This lower bound is based on the lower bounds of the IPPS, and the job-shop scheduling problem [1,2]. For each resource

$r \in R$, we find the sum of processing times of all jobs that must be processed by resource r and add the minimum *head* and *tail* of the jobs $j \in J_r$. This gives a lower bound on the makespan of jobs scheduled on resource $r \in R$. The minimum *head time* of resource $r \in R$ is defined as the minimum starting time of any job $j \in J_r$. The minimum *tail time* of resource $r \in R$ is defined as the minimum remaining processing time at all jobs processed on r. We define variable E_r^1 (resp. E_r^2) to be the earliest head (resp. tail) time on resource $r \in R$. This gives the following lower bound on the makespan where $c_b(r)$ is the minimum starting time of the first job of branch b on resource $r \in R$.

$$c_{\max} \geq E_r^1 + E_r^2 + \sum_{I \in \mathcal{I}} \sum_{b \in B(I)} c_b(r) x_b \qquad \forall r \in R \tag{7}$$

Finding the minimum head/tail time on resource $r \in R$ involves additional constraints and variables. In this formulation, Y_r is a binary variable indicating if any job is assigned to resource $r \in R$. Binary variable $Q_{r,b}^h$ indicates whether the branch b on resource $r \in R$ is picked as the earliest head or tail branch of resource r ($h \in \{1, 2\}$). For a large number, we use $M = \sum_{I \in \mathcal{I}} \sum_{b \in B(I)} c_b$ as the sum of all branch cost as an upper bound on the makespan. The complete master problem is shown in (MP). For ease of notation, we let B_r be the set of all branches that contain a job that must be processed by resource $r \in R$.

$$\min \quad c_{\max} \tag{MP}$$

$$\text{s.t.} \quad \sum_{b' \in N[b]} x_{b'} = 1 \qquad \forall I \in \mathcal{I},\, b \in B(I) \tag{8}$$

$$c_{\max} \geq E_r^1 + \sum_{I \in \mathcal{I}} \sum_{b \in B} c_b(r) x_b + E_r^2 \qquad \forall r \in R \tag{9}$$

$$Y_r \geq x_b \qquad \forall r \in R,\, b \in B_r \tag{10}$$

$$\sum_{b \in B_r} Q_{r,b}^h = Y_r \qquad \forall r \in R,\, h \in \{1, 2\} \tag{11}$$

$$Q_{r,b}^h \leq x_b \qquad \forall r \in R,\, b \in r,\, h \in \{1, 2\} \tag{12}$$

$$E_r^1 \geq \sum_{\beta \prec b} c_\beta x_\beta + c_b(r) x_b - M(1 - Q_{r,b}^1) \qquad \forall r \in R,\, b \in B_r \tag{13}$$

$$E_r^2 \geq \sum_{\beta \succeq b} c_\beta x_\beta - c_b(r) x_b - M(1 - Q_{r,b}^2) \qquad \forall r \in R,\, b \in B_r \tag{14}$$

$$Y_r, Q_{r,b}^h \in \{0, 1\} \qquad \forall r \in R,\, b \in r,\, h \in \{1, 2\} \tag{15}$$

$$E_r^h \geq 0 \qquad \forall r \in R,\, h \in \{1, 2\} \tag{16}$$

$$x_b \in \{0, 1\} \qquad \forall I \in \mathcal{I},\, b \in B(I) \tag{17}$$

$$c_{\max} \geq 0 \tag{18}$$

The variable Y_r is only equal to 1 if there exists a job on the resource by Constraint (10). Constraint (11) ensures that at most one branch from each resource is selected and Constraint (12) ensures that only selected branches can be considered for the earliest head/tail time. The selected branch is then used to find the earliest head/tail time E_r^h on each resource $r \in R$ in Constraint (13) and Constraint (14). E_r^h is minimized since it is positively correlated with makespan $c_{\max}$.

We add benders cuts to the master problem. Since the master problem selects the branches, different to the ICP, the CP subproblem does not have any optional jobs and we can use the pure scheduling formulation (SP). We use standard monotone cuts on the selected branches as described by Hooker [8]. Here $\overline{x}$ represents all variables x_j for which $\overline{x}_j = 1$, $v^*(\overline{x})$ is the optimal makespan value for the schedule induced by $\overline{x}$, and $\underline{v}$ is the global lower bound on the makespan. The branches selected by $\overline{x}$ is indicated as $B(\overline{x})$.

$$c_{\max} \geq v^*(\overline{x}) - (v^*(\overline{x}) - \underline{v}) \sum_{b \in B(\overline{x})} (1 - x_b) \tag{19}$$

In this monotone cut, only one specific configuration is eliminated. Ideally we would like to eliminate multiple configurations through a single cut. To do this, we find a lower bound on the makespan for a subset of process instances that is greater than the global lower bound. We select a subset $\mathcal{I}' \subseteq \mathcal{I}$ with branch selection $B(\overline{x}') = (\cup_{I \in \mathcal{I}'} I) \cap B(\overline{x})$. If the makespan lower bound for the branches $B(\overline{x}')$ is greater than the global lower bound, we can add this as a strengthened cut. This means all possible configurations can be eliminated for any $I \notin \mathcal{I}'$. Solving a CP for all subsets $\mathcal{I}'$ is computationally intensive, so we consider a lower bound that can be found efficiently. For this lower bound, we compute the minimum resource usage starting after every integer time t. Let $c_r^I(t)$ indicate the cost of all jobs that must schedule at or after t on resource $r \in R$ for process instance $I \in \mathcal{I}'$. This gives the lower bound in (20) for the branches $B(\overline{x}')$.

$$v^*(\overline{x}') \geq \max_{r \in R}(\max_{t \in T}(t + \sum_{I \in \mathcal{I}'} c_r^I(t))) \qquad \forall \mathcal{I}' \subseteq \mathcal{I} \tag{20}$$

The subproblem only schedules the jobs J that have been selected by the master problem. Different to the integrated CP, the subproblem no longer has to enforce the presence of individual jobs, since all must be present.

$$\min \max_{j \in \cup_{I \in \mathcal{I}} J(I)} (\text{EndOf}(Task_j)) \tag{SP}$$

$$\text{s.t. } Task_j = \text{IntervalVar}(c_j) \qquad \forall I \in \mathcal{I},\, j \in J(I) \tag{21}$$

$$\text{NoOverlap}(Task_i, Task_j) \qquad \forall r \in R,\, i, j \in J_r \tag{22}$$

$$\text{EndBeforeStart}(Task_i, Task_j) \qquad \forall I \in \mathcal{I},\, i, j \in J(I),\, i \prec j \tag{23}$$

Constraint (21) defines the interval variables with length c_j. Constraint (22) ensures that each resource processes at most one job at any time. Constraint (23) enforces the precedence constraints of the selected jobs. (SP) has as objective the makespan of the schedule.

4 Application in Offline and Online Scenarios

We test three optimization approaches for the configuration and scheduling problem of RA-PSTs. Following Sect. 3 we use (ICP) as integrated configuration and scheduling CP. LBBD uses (MP) and (SP). As comparison, we combine (Config) and (SP), which solves configuration and scheduling sequentially (Config+CP).

We test the approaches in an offline and an online setting as shown in Fig. 4. In the offline application, all process instances to be scheduled are known beforehand and are configured and scheduled simultaneously. In the BPM-driven online setting, process instances arrive over time. Each process instance is configured and scheduled independently. The solver uses current schedule information to find a configuration for the arriving instance.

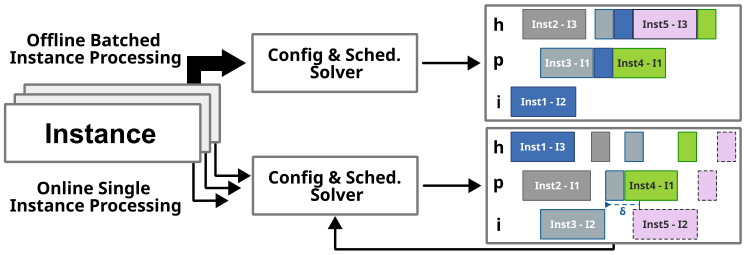

Fig. 4. Visualization of the offline and online scheduling problem. Parameter δ represents possible right-shift of already scheduled tasks. "Inst5-I2" is the process instance to be configured & scheduled next in the online setting.

All approaches are implemented in Python using IBM CPLEX 22.1 for CP. The *(optional) interval* and *sequencing* variables available in the CP Optimizer allow for intuitive modeling of the scheduling problem [11]. The MIP is implemented in Gurobi 12.0 by Gurobi[1]. Experiments were run on a computer with Core™ i7-1165G7 CPU @ 2.80 GHz. The quality of a solution obtained by a solver is defined by a lower and an upper bound. The upper bound is the objective value of the best feasible solution found. The lower bound is a value that is no bigger than the objective value of any feasible solution and can be found through a relaxation of the problem. When the upper and lower bounds are equal, a solution is optimal.

4.1 Offline Scheduling:

We test ICP, LBBD, and Config+CP on an offline test set. Table 1 gives an overview of the used test sets. We design RA-PSTs with growing task numbers for the process model, namely 2, 5, and 10 Tasks. The 2-task process reflects the example presented in Sect. 2. Additionally, the problem size is increased by

[1] Code: https://github.com/Schlixmann/Instance_config_and_schedule/.

using process models with more branches, i.e., more flexibility on which resource can execute a task. For each test set, 8 process instances are scheduled, and all instances are released at the same time, i.e., at 0 time units. In test sets 7 and 8, different RA-PSTs are chosen randomly, i.e., for the first process instance, a different RA-PST is used than for the second instance. In this setting, the number of branches and also possible configurations per instance vary. Each test set is run for each approach for 7200s. For the LBBD we run the (SP) for 5 s.

Table 1. Description of test sets, since multiple RA-PSTs are used in 7, 8, branch and configuration per RA-PST is not given

id	unique RA-PST	tasks	resources	branches	configs p. Instance
1	1	2	4	5	4
2	1	5	5	7	4
3	1	5	5	24	32
4	1	10	5	20	1.024
5	1	10	5	37	442.368
6	1	10	5	29	39.366
7	5	10	5	-	-
8	5	10	5	-	-

Table 2 presents the computational results for the offline use case. We can see that when configuration and scheduling are separated (Config+CP), an optimal schedule can be found for all tests (LB=UB). Since this approach schedules the same configuration for each instance, as expected, Config+CP does not find a better upper bound than ICP and LBBD. For the small test sets 1, 2, and 3, ICP finds the optimal solution in all cases. LBBD finds the best upper bound for test sets 1, 2, and 3 and the optimal lower bound for test sets 1 and 3. For test set 2 LBBD fails to find a better lower bound than ICP. For larger instances in test sets 4–8, ICP always finds the best upper bounds but struggles to find meaningful lower bounds. Here, the LBBD approach finds much better lower bounds but cannot always find upper bounds as good as the ICP. ICP and LBBD perform best for test sets 7 and 8. Specifically, the LBBD approach benefits from the differentiation of RA-PSTs. This is promising for integrating the optimization approaches into scenarios with a multitude of simultaneously running processes and process variants.

4.2 Online Scheduling and Rescheduling

The online setting refers to process instances executed by a process engine that are typically deployed based on their arrival [7]. In the following, we apply CP, LBBD, and Config+CP in this "online" fashion in order to test their readiness

Table 2. Lower Bound (LB), Upper Bound (UB), Gap to best LB among all solution approaches and computing time, Config+CP serves as UB reference

	ICP				LBBD				Config+CP			
	LB	UB	Gap best LB	Time s	LB	UB	Gap best LB	Time s	LB	UB	Gap best LB	Time s
1	**115**	**115**	**0.0%**	**0.41**	**115**	**115**	**0.0%**	2032.9	165	165	30.3%	0.02
2	**63**	**63**	**0.0%**	**0.16**	57	**63**	**0.0%**	7200	84	84	25.0%	0.02
3	**102**	**102**	**0.0%**	24.27	**102**	**102**	**0.0%**	**0.05**	144	144	29.2%	0.01
4	21	**94**	**23.4%**	7200	**72**	103	30.1%	7200	108	108	33.3%	1.04
5	14	**81**	**11.1%**	7200	**72**	94	23.4%	7200	107	107	32.7%	0.02
6	21	**101**	**23.8%**	7200	**77**	112	31.2%	7200	163	163	52.8%	0.06
7	29	**94**	**6.4%**	7200	**88**	**94**	**6.4%**	7200	139	139	36.7%	0.02
8	29	**94**	**4.3%**	7200	**90**	95	5.3%	7200	145	145	37.9%	0.01

for the online setting. Instances arriving over time add uncertainty to the problem at hand. The flexibility in the RA-PST enables a reaction to counter this uncertainty. To show the effect of this flexibility, we also use a direct allocation approach that operates directly on the RA-PST as a comparison. It resembles the ad-hoc approaches often used in BPM by assigning tasks to resources at task release. While being the most reactive allocation approach, it offers no stable execution plan for a process instance, and tasks lying in the past can not be deleted. The greedy allocation allocates one task at a time. Once the execution of a task is finished, the next task is allocated by choosing the branch with the earliest finish time.

To harden scheduling approaches against uncertainty at runtime, rescheduling is a common approach [15]. We balance complexity and enable a reaction to newly arriving instances for the optimization approaches by allowing to right-shift already scheduled tasks by factor δ as a rescheduling measure. A re-configuration of planned instances is not possible.

To test the online set-up, multiple RA-PSTs have been designed. Due to the single process instance usage, the approach can handle bigger process models. We test processes with 10, 20, and 30 tasks and have five resources. From an RA-PST perspective, up to 4 branches per task are allowed. For each process size, 10 test sets are used. Again, 2 test sets consist of different RA-PSTs. Factor δ is set to the length of one average task length in the RA-PST, and arrival times follow a Poisson distribution; the time between each instance arrival is an exponential distribution with a scaling factor of one average task length [12]. The time needed to find a solution is important in the online setting. Integrated into a BPM System, the process engine keeps running processes in parallel to the scheduling algorithm. We allow for 100s of runtime for the configuration and scheduling of a single instance. As a comparison oracle, we let the ICP solve the problem globally for all process instances simultaneously. We use the objective found after 100s of runtime as a comparison.

Table 3 shows the median relative deviation for each allocation type from the oracle. For processes with 10 Tasks, the integrated ICP with rescheduling finds an optimal solution for each process instance in each test set. The optimum for the bigger test sets cannot be found in all cases within 100s. Through more detailed data analysis, we found that in many cases, the ICP struggles to find the best schedule for the first instance in a test set but configures the following process instances optimally. This could easily be overcome by using the (Config) formulation to schedule the first instance. The impact of rescheduling is substantial. The deviation from the oracle rises if rescheduling is prohibited. As in the offline setting, for the Config+CP approach the found solution is worse than the other approaches due to the lack of configuration combination. For the overall makespan within each test set, the greedy allocation can outperform the ICP for the processes with more tasks where the higher level of reactivity proves helpful. The LBBD approach does not perform well in the online setting. In the majority of cases, the optimum cannot be found.

Table 3. Performance of the different approaches in the online setting

	10 Tasks			20 Tasks			30 Tasks		
	Deviation	Optimal	Percentage	Deviation	Optimal	Percentage	Deviation	Optimal	Percentage
Oracle	0.0	-	-	0.0	-	-	0.0	-	-
Resched. ICP	**22.1**	80/80	100.0%	36.9	77/80	96.2%	75.1	63/80	78.8%
Resched. LBBD	26.2	43/80	53.8%	59.4	12/80	15.0%	102.3	10/80	12.5%
Greedy	30.0	-	-	**26.4**	-	-	**58.4**	-	-
No. Resched. ICP	38.3	80/80	100.0%	73.1	78/80	97.5%	135.4	74/80	92.5%
No. Resched. LBBD	38.7	34/80	42.5%	77.6	11/80	13.8%	127.6	10/80	12.5%
Resched. Config+CP	58.5	80/80	100.0%	95.0	80/80	100.0%	130.6	80/80	100.0%

The core idea of RA-PSTs is to provide the option of flexibility for resource allocation. To better analyze the different solution approaches, we compare the performances based on the flexibility provided by the RA-PST at design time. We adopt the entropy measure proposed in [16], which, in our proactive case, represents the flexibility of configurable PSTs. A higher entropy represents a higher flexibility since more similar branches are available for a decision. The entropy of an RA-PST is calculated as the mean entropy over all PST tasks. The entropy for a single task t with a set of branches B and the corresponding costs of a branch c_b can be calculated by using the Sum of all inverted costs: $\sigma = \sum_{b' \in B} \frac{1}{c_{b'}}$ The entropy for one task is then defined as:

$$H_t = -\sum_{b \in B} \frac{1}{c_b \cdot \sigma} \log(\frac{1}{c_b \cdot \sigma}) \qquad \forall b \in B \tag{24}$$

Figure 5 shows the relative makespan deviation for each test over the entropy normalized over all test sets. We can see that the greedy allocation performs best

when the entropy, and therefore flexibility, is high. The high reactivity of the ad-hoc approach facilitates the flexibility the best. For RA-PSTs with lower entropy, the greedy allocation cannot find as good objective values as the rescheduling ICP. The degree of flexibility given in a test set barely impacts the objective value if no rescheduling is allowed. We suspect that rescheduling ICP could perform better if the δ parameter is increased.

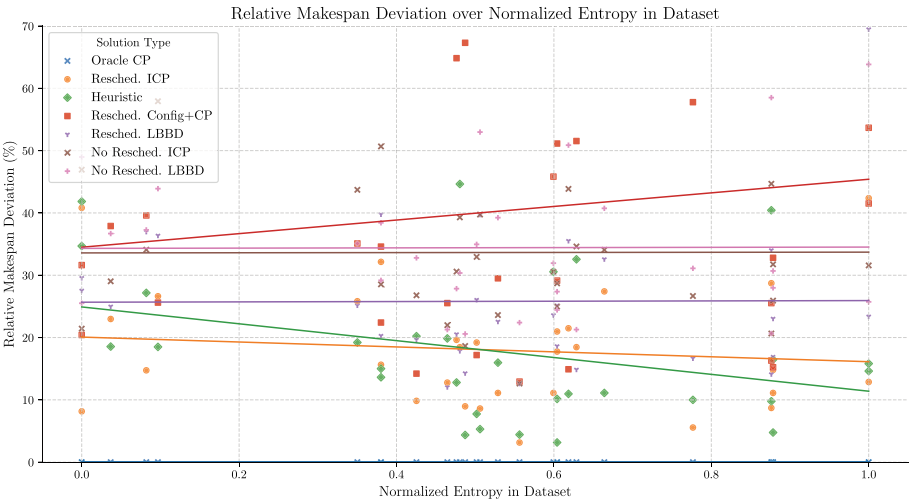

Fig. 5. Relative makespan deviation by normalized entropy over all RA-PSTs. Lines represent the trend over all data points for each solution approach.

5 Related Work and Discussion

The problem of (optimal) business process scheduling has been studied in the BPM community with different focus areas on the combination of BPM and OR methods. [6] proposes process fragmentation to deal with uncertainties in process models and uses Answer Set Programming to solve the scheduling problem. The approach can be used offline and in a quasi-online approach. In [7] different Answer-Set-Programming solvers are used to solve a scheduling problem and compared. The problem complexity is defined by the parallelity of the business process and the number of tasks. The maximum problem size contains 64 tasks and 32 resources and uses only a single process instance with parallelities. [18] proposes a method to learn scheduling models from timed petri nets and automatically solves these scheduling models with the help of CP. As stated in [18], most approaches for scheduling of timed petri nets focus on basic scheduling elements like precedence constraints and resources with limited capacities. When looking at scheduling of different alternatives for a process from the OR perspective, [19,20] propose metaheuristics to deal with the Resource Constrained

Project Scheduling Problem with arbitrary subgraphs. These problems allow for different execution alternatives. In distinction to the aforementioned RA-PST problem, alternatives are not linked to the resources executing them.

Discussion: Besides precedence constraints, process models can also contain exclusive choices, loops, and parallels as part of the control flow. In accordance with [1,10], we deal with purely sequential processes in which the variability is created through the branches of the RA-PST. Adding exclusive choices requires online process execution data to deal with the added uncertainty and is therefore not feasible for this study [6]. Dealing with uncertainty and online schedules is also a separate research field in OR [5,15]. By scheduling multiple process instances, we already deal with parallelities in terms of the scheduling problem itself. Comparable approaches in BPM create parallelity by scheduling a single instance with parallel gateways [7]. While the presented solution models can be easily adapted to allow for parallel gateways within a process instance, it would further increase the computational complexity of the problem.

Literature on the IPPS limits the number of alternative process plans (here, configurations of an instance) that are given to the optimization algorithm. In [9], a preselection of only four possible process plans for a part type (process instance) is done by choosing the cheapest options. This lowers the complexity of the problem drastically. For the BPM community, larger instances, such as those used for our evaluation, are typical, and the RA-PST naturally leads to bigger problem spaces. This might be an explanation of why the LBBD approach did not always find the optimum. For future work, better strengthened cuts can improve the performance of LBBD. [1] proposes additional cuts, which could lead to the deletion of optimal solutions. We refrain from using these cuts on our problem.

6 Conclusion

We presented three scheduling methods for the configuration and scheduling problem on RA-PSTs. The ICP formulation, which handles configuration and scheduling combined, finds good upper bounds for any problem in our experiments. Yet the approach struggles to find meaningful lower bounds, once the problem size grows. In contrast, the LBBD finds better lower bounds for larger problem sizes but struggles to find the good upper bounds the ICP can deliver in a timely manner. In the online setting, the ICP finds the optimal solution for single instances in most cases. Due to the reduced problem size in this setting also bigger process models with many tasks and branches become solvable. The LBBD approach is struggles in the online setting. For the BPM community, this work opens a path to using combinatorial optimization methods for configurable, executable process models. We show that the ICP formulation can be used in offline and online settings. In future work, introducing runtime data from an execution engine to online decision-making could be used to further improve both configuration and scheduling at process instantiation. Tying scheduling closer to

business process execution will help to make BPM more proactive not only from a control flow but also from a resource perspective.

Acknowledgments. This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – GRK2201 - Projektnummer – 277991500.

References

1. Barzanji, R., Naderi, B., Begen, M.A.: Decomposition algorithms for the integrated process planning and scheduling problem. Omega **93** (2020)
2. Carlier, J., Pinson, E.: An algorithm for solving the job-shop problem. Manage. Sci. **35**(2), 164–176 (1989)
3. Demange, M.: A note on the approximation of a minimum-weight maximal independent set. Comput. Optim. Appl. **14**(1), 157–169 (1999)
4. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
5. Gupta, D., Maravelias, C.T., Wassick, J.M.: From rescheduling to online scheduling. Chem. Eng. Res. Des. **116**, 83–97 (2016)
6. Havur, G., Cabanillas, C.: History-aware dynamic process fragmentation for risk-aware resource allocation. In: OTM, pp. 533–551 (2019)
7. Havur, G., Cabanillas, C., Polleres, A.: Benchmarking answer set programming systems for resource allocation in business processes. Expert Syst. Appl. **205** (2022)
8. Hooker, J.: Logic-Based Benders Decomposition: Theory and Applications. Springer (2023)
9. Jain, A., Jain, P., Singh, I.: An integrated scheme for process planning and scheduling in FMS. Int. J. Adv. Manuf. Technol. **30**, 1111–1118 (2006)
10. Naderi, B., Begen, M.A., Zaric, G.S.: Type-2 integrated process-planning and scheduling problem: reformulation and solution algorithms. Comput. Oper. Res. **142**, 105728 (2022)
11. Naderi, B., Ruiz, R., Roshanaei, V.: Mixed-integer programming vs. constraint programming for shop scheduling problems: new results and outlook. INFORMS J. Comput. **35**(4), 817–843 (2023)
12. Nie, L., Gao, L., Li, P., Li, X.: A GEP-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates. J. Intell. Manuf. **24**(4), 763–774 (2013)
13. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems, 3rd edn. Springer (2008)
14. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies. Springer, Heidelberg (2012)
15. Sabuncuoglu, I., Goren, S.: Hedging production schedules against uncertainty in manufacturing environment with a review of robustness and stability research. Int. J. Comput. Integr. Manuf. **22**(2), 138–157 (2009)
16. Saidi, M., Tissaoui, A., Benslimane, D., Faiz, S.: Uncertainty measurement of a configurable business process. Syst. Eng. **26**(2), 199–215 (2023)
17. Schumann, F., Rinderle-Ma, S.: Optimizing resource-driven process configuration through genetic algorithms. In: BPM, vol. 14940, pp. 3–20 (2024)
18. Senderovich, A., Booth, K.E.C., Beck, J.C.: Learning scheduling models from event data. In: ICAPS, pp. 401–409 (2019)

19. Servranckx, T., Coelho, J., Vanhoucke, M.: A genetic algorithm for the resource-constrained project scheduling problem with alternative subgraphs using a Boolean satisfiability solver. Eur. J. Oper. Res. **316**(3), 815–827 (2024)
20. Servranckx, T., Vanhoucke, M.: A tabu search procedure for the resource-constrained project scheduling problem with alternative subgraphs. Eur. J. Oper. Res. **273**(3), 841–860 (2019)
21. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and more focused control-flow analysis for business process models through SESE decomposition. In: ICSOC, pp. 43–55 (2007)