Optimizing Resource-Driven Process Configuration through Genetic Algorithms

Felix Schumann[0009−0006−5431−1922], Stefanie Rinderle-Ma[0000−0001−5656−6108]

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

Abstract. In order to optimize the efficiency of operations in organizations, the control flow of business processes and the resources allocated to process tasks have to be considered in an intertwined way. In real-world process scenarios, resources might even manipulate the control flow, e.g., if the allocation of a certain resource to one task renders the execution of another task superfluous. Hence, we advocate to equip resources with change patterns, resulting in process configuration at instance level. This raises the challenge of determining executable process configurations with valid and, at the same time, optimal resource allocations w.r.t. some optimization goal. To this end, we introduce and utilize the concept of the Resource-Augmented Process Structure Tree (RA-PST) with insert, replace, and delete patterns for resources. The RA-PST combines the variability of configurable process models with optimization-focused resource allocation modeling. It is shown how the validity of the resource allocation and the soundness of the resulting process instance can be checked based on the constructed RA-PST. For the combinatorial optimization problem of resource allocation, we adopt a genetic algorithm and test it on five different sets of resources. The results showcase the effectiveness of focusing on resource optimization during business process modeling and demonstrate how an optimal configuration can be achieved, i.e., the genetic algorithm finds (near-) optimal solutions, especially when heuristics are not able to handle the additional complexity.

Keywords: Process Configuration · Resource Allocation · Process Structure Tree · Genetic Algorithm

1 Introduction

While the focus on the structural soundness of process models is an important enabler for automation, it limits itself to the control-flow perspective [5]; the process models can reflect the logical order of operations in a formalized way, but often miss the impact of involved actors (resources) on the actual process execution [7]. In contrast, optimization models from Operations Research (OR) focus on the distribution of operations over a set of resources in a system. These approaches are considered universal and applied in many business areas, such as manufacturing, transportation, and health care. Its practitioners claim that

their optimization methods have substantially impacted global productivity over the last decades [14]. The idea that process optimization can be achieved purely by focusing on control-flow soundness is a misconception [5]. As we can see from OR applications, optimization must also consider the involved physical resources that drive the execution of a process [6].

In recent years, the interest in process optimization has risen, and many scholars advocate the combination of BPM and OR methods [5,16,18,24]. However, a structural process framework is missing that combines the strengths of both fields. On the one hand, the structural soundness of a process model must be ensured to enable process automation. On the other hand, it must be able to incorporate the optimization of resource allocations for a process or system as known from OR. Based on OR literature, we argue that such a framework must enable variability of the control-flow and set a focus on executing resources [14].

Hence in this work, we propose a process model that is inspired by optimization models from OR. The approach allows resources to alter the control-flow in a structured manner, if they become a participant in the process (i.e., are allocated to a task). This resembles the modeling structure of known OR problems, e.g., the parallel machine scheduling problem [8]. The proposed approach enables control-flow variability, a multi-perspective view, and ensures structural soundness. Resources are allowed to insert, replace, and delete tasks in the control-flow to create variability. The optimal operational process performance can then be achieved through sophisticated resource allocation methods known from OR. To the authors' knowledge, no current modeling approach enables this interdependency between resource perspective and control-flow perspective.

The formalized model is based on a Process Structure Tree (PST). This PST is enhanced with information on possible resource allocations and the changes they will introduce to the control-flow. To do so, we extend the tree structure by resource, resource profile, and change pattern nodes. We show how this structure can be used for i) validity verification of the resulting process configuration at process instance level (including structural soundness and valid resource allocation) and ii) optimization of the resource allocation of the full process by adopting a genetic algorithm to solve the resource allocation problem.

Based on the idea of Resource-Driven Process Manipulation for a single task as described in [23], this work fully formalizes the RA-PST and provides an algorithm to create an RA-PST with allocation information for a full process model. This enables us to find a (near-) optimal allocation and configuration for a complete process instance by considering the dependencies between the different allocations. The newly introduced "delete" change pattern requires the consideration of both valid and invalid branches to be able to find the best global allocation. Therefore, we show the requirements needed to find a valid configuration for an instance. We provide a genetic algorithm to find global solutions to the resulting combinatorial optimization problem and compare it to the best branch heuristic, which only finds local solutions through the sequential allocation of single tasks without considering the dependencies between different allocations.

Section 2 formalizes the RA-PST. Section 3 shows how process instance models are configured based on the RA-PST and how the validity of a process configuration can be verified based on the RA-PST. Section 4 describes how to design a genetic algorithm to solve the allocation problem posed by the RA-PST. Section 5 gives an overview of related work and Sect. 6 concludes the paper.

2 Augmenting Process Structure Trees with Resource Allocations – The Process Model Level

Let the control flow of a process P consist of a set of tasks N and a set of directed edges E, i.e., $P := (N, E)$. In this work, we assume that P follows the meta model of the Process Structure Tree (PST) [25]. The structure of a PST is defined as the composition of multiple Single-Entry-Single-Exit (SESE) fragments. By adapting Theorem 2 in [25] to the PST *the PST is sound if and only if all child fragments are sound and the PST that is obtained by replacing each child fragment with an activity is sound.* Moreover, the task-based structure of the PST allows us to implement the approach in a TPST-like manner as described in [26]. Figure 1 shows a running example process model in BPMN (`bpmn.org`) notation (a) and the corresponding PST in (b).

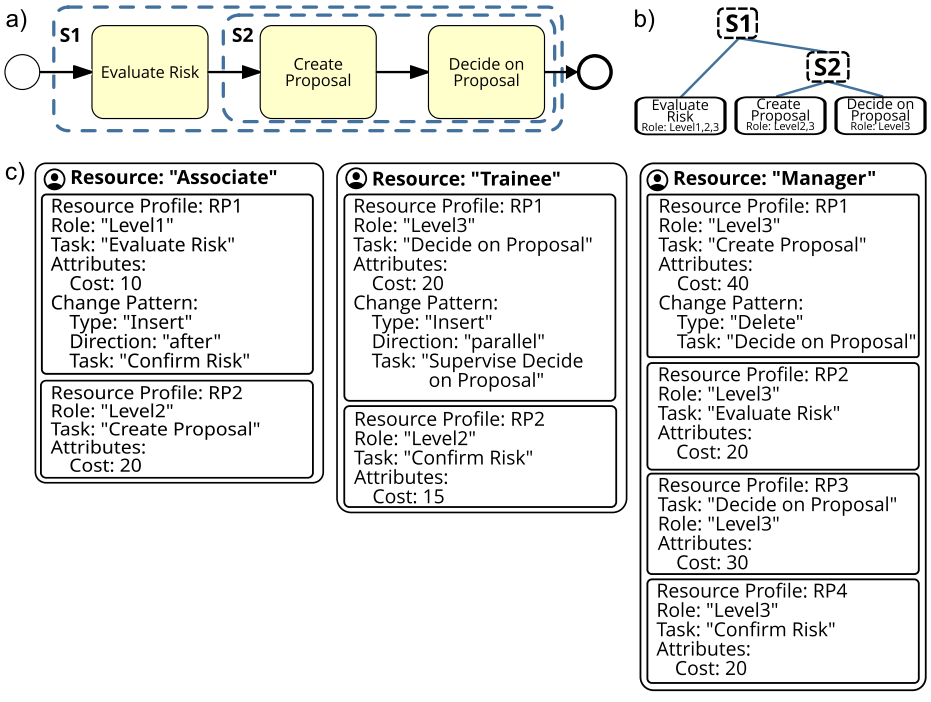

Fig. 1: Running example: a) Process in BPMN; b) Process as PST; c) Resource description with three resources and resources profiles

Next, we define resource profiles which form the basis for resource allocation and resource-driven manipulation of the PST.

Definition 1 (Resource Profile, adapted from [23]). *Let $\mathcal{T}$ be the set of all process tasks, $\mathcal{R}$ be the set of resources, Role be the set of roles, $\mathcal{A}$ be the set of attributes, and $CP = \{Insert(x, y, before|after|parallel), Replace(x, y), Delete(x)\}$, $x, y \in \mathcal{T}$ a set of change patterns [23,19]. A resource profile rp is defined as $rp := (r, role, t, Attr, cp)$ with*

- *$r \in R$ is a resource*
- *$role \in Role$ a role*
- *$t \in \mathcal{T}$*
- *$Attr \subseteq \mathcal{A}$ a set of attributes*
- *$cp \subseteq 2^{CP}$ a set of change patterns*

As can be seen from Def. 1, a resource profile *rp* refers to a resource r and its role *Role* which, in turn, is linked to a process task t to achieve a role-based allocation structure [20]. Typically, process tasks define one or more roles that specify authorization to perform the task. Implicitly, we assume that the role specified at the task corresponds to the role in the resource profile. This might seem redundant at first glance, but is necessary for future extension for role-based change pattern inheritance.

rp can manipulate the process based on a selection of change patterns as provided in literature [19,23], i.e., inserting a task x before, after, or parallel to task y, replacing a task x by another task y, and deleting x. In the sequel, the correct application of the change patterns on process models will be ensured by the validity verification as presented in Sect. 3.

To enable the optimization towards a process-specific optimization goal, different attributes such as costs or execution time can be part of a resource profile.

Figure 1c) depicts three resources, i.e., `Associate`, `Trainee`, and `Manager`. For `Associate`, for example, two resource profiles exist. RP1 has role `Level1` and refers to task `Evaluate Risk`. If `Associate` is allocated with RP1, the process is manipulated by inserting task `Confirm Risk` after `Evaluate Risk`.

The main question is how to allocate the resources given the set of resource profiles such that the resource allocation is *valid* and *optimal* w.r.t. some optimization target, for example, minimizing the cumulative costs of all allocations. As stated in literature [20] and transferred to the terminology of this paper, resource allocations are *valid* iff they do not refer to resources that are not present in any resource profile and the allocation can be satisfied by at least one resource in the set of resource profiles. As opposed to existing approaches, checking validity might become more challenging due to the ability of resources to manipulate the PST, especially for delete operations.

In the following, we present the concept of the **Resource-Augmented PST (RA-PST)**, i.e., a PST augmented with the resource allocations defined in the resource profiles. The benefit of this augmentation is that all interactions between resource allocations and the associated change operations with the PST

and among each other can be systematically analyzed and **invalid** resource allocations can be detected. Moreover, the costs of each resource allocation can be determined and serve as input for the optimization problem.

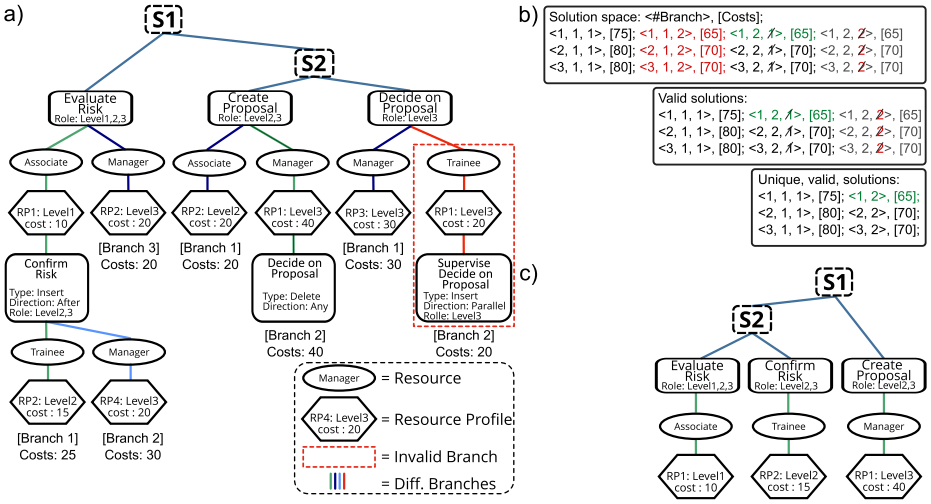

Fig. 2: a) RA-PST resulting from process and resource description in Figure 1; b) Possible branch configurations for RA-PST with costs. Invalid task/configuration red, best configuration green, duplicate configuration grey, to be deleted task crossed out; c) Configured RA-PST$_{instance}$ model for optimal solution <1,2> with allocated resources, derived from a) through change pattern application

The basic idea of augmenting the PST of process P with the allocation information of the resource profiles and resources has been introduced in [23] for insert and replace change patterns. This work, in addition, considers the delete change pattern, which adds another level of complexity to the resource allocation problem. Extending [23], the delete change pattern creates interdependencies between branches, which require the creation of an RA-PST for the full process model instead of creating an allocation tree for only one task. This enables the (optimal) configuration of a whole instance instead of looking at the local allocation for one task. The delete change pattern influences the presence of tasks for other resource allocations. Invalid branches can not be pruned from the RA-PST and must be considered in the configuration step in order to find an optimal solution.

The RA-PST is constructed as described in Alg. 1 and takes the PST and the set of associated resource profiles as input. Figure 2a represents the full RA-PST for the given example at the process model level. After selecting a combination of branches for each leaf in the original PST, the change patterns contained in the branches of the RA-PST are applied to the original PST, configuring the corresponding RA-PST instance model RA-PST$_{instance}$ (cf. Figure 2c). This RA-

$\text{PST}_{instance}$ can be executed by a process engine. The configuration of the RA-$\text{PST}_{instance}$ model is presented in Sect. 3, together with the validity verification.

Algorithm 1: build_RA-PST_tree

```
   input  : PST for P=(N,E), set of resource profiles RP with
            rp := (r, role, t, Attr, cp) ∈ RP
   output: RA-PST
 1 excluded_tasks := ∅
 2 for task in N do
   |  //Add allocation tree as child to each task node
 3 |  excluded_tasks := ∅
 4 |  build_allocation_tree(task, RP))
 5 end
 6 Function build_allocation_tree(task, RP):
 7 |  for rp in RP do
 8 |  |  if rp.t == task & rp.role == task.role then
 9 |  |  |  r_node := add_child(t, {rp.r})
10 |  |  |  rp_node := add_child(r_node, {rp.role,rp.Attr})
11 |  |  |  for ct in cp do
12 |  |  |  |  if ct.x in excluded_tasks then
13 |  |  |  |  |  remove_child(rp_node)  //prevent cyclic allocation
14 |  |  |  |  |  continue
15 |  |  |  |  end
16 |  |  |  |  ct_node := add_child(rp_node, {rp.ct})
17 |  |  |  |  if ct.type == delete then
18 |  |  |  |  |  continue
19 |  |  |  |  end
20 |  |  |  |  if ct.type == replace ∨ insert then
21 |  |  |  |  |  excluded_tasks.append(ct.x)
22 |  |  |  |  |  build_allocation_tree(ct.x, RP)
23 |  |  |  |  end
24 |  |  |  end
25 |  |  |  if r_node.children == ∅ then
26 |  |  |  |  remove_child(r_node)
27 |  |  |  end
28 |  |  end
29 |  end
30 |  return task
```

For constructing the RA-PST, the basic principle is to assign to each task (aka leaf in the PST, Line 1 in Alg. 1), all resource allocations that correspond to the task specified in the resource profile in order to meet the following requirements:

1. The RA-PST represents all possible combinations of resource allocations and change patterns.
2. The resulting construct is an (augmented) PST based on which the validity and costs of a branch can be determined automatically.
3. Applying the change patterns of a combination of branches configures a structurally sound, executable RA-PST$_{instance}$ model for which the validity and cumulative costs of the resource allocations can be determined.

In order to meet Req. 1, the resource allocations in the RA-PST are expressed by branches/trees. To this end, the node set of the original PST for P=(N,E) is augmented with nodes of three additional node types reflecting i) the resource (*r_node*, Line 3), ii) the resource profile (*rp_node*, Line 9) and iii) the change patterns (*ct_node*, Line 15). The nodes are then arranged into one branch in an alternating way, starting from the resource node assigned to a task, then generating and appending the resource profile node, and finally appending the change pattern node. The semantics of this construction is that a task is assigned a resource through a resource profile which can hold specific change patterns. The change patterns, in turn, might refer to tasks that might be referenced by other resource profiles, hence resulting in the addition of further resources. The construction of the resource allocation branches is described in function `build_allocation_tree` (Lines 5 – 26, cf. Alg. 1) which generates the respective nodes and uses them to construct a branch. Note that function `build_allocation_tree` is called recursively to append resource allocations based on the specified change patterns.

Based on the PST and resource profiles depicted in Fig. 1, the RA-PST shown in Fig. 2a is built. For task `Evaluate Risk`, two resources can be allocated, i.e., `Associate`, and `Manager` (depicted as oval-shaped nodes). Hence, two resource nodes are generated and added as children to the task node. At the same time, the corresponding resource profile nodes `RP1` in the context of `Associate` and node `RP2` in the context of `Trainee` are generated and appended as child nodes to the respective resource nodes (depicted as diamond-shaped nodes). Then it is checked which of the resource profiles specifies a change pattern, i.e., `RP1` for `Associate` in this case. Hence, a change pattern node is generated and appended to the resource profile node (depicted as rounded rectangle). The change pattern inserts tasks `Confirm Risk`. `Confirm Risk` is a task in resource profile `RP2` for resource `Trainee` and `RP4` for `Manager`. Hence, two resource nodes are created and appended to the change pattern node, together with their child resource profile nodes. For these resource profiles there is no more change pattern specified. Hence, the construction of the resource allocation branches for task `Evaluate Risk` stops at this point. The RA-PST also contains the costs per resource allocation branch (bottom) by summing up all costs stored in the resource profile nodes along the branch (cf. Figure 2a).

Allocating resources and subsequently configuring the process model based on the RA-PST constitutes the transition from the process model to the process instance level (cf. Req. 2 & 3). In Fig. 2a, for example, the RA-PST at the process model level is configured based on change pattern resolution and resource

allocation into the RA-PST$_{instance}$ at the process instance level depicted in Fig. 2c, i.e., into an executable process with resource allocation. In Sect. 3, we present process configuration based on resolving change patterns in resource profiles as contained in the RA-PST, together with the *validity* of the resulting RA-PST$_{instance}$ (cf. Req. 3). Further we show how to to allocate resources based on finding the best combination to optimize costs.

3 Process Configuration – The Process Instance Level

In this section, we show how to configure a valid process instance model RA-PST$_{instance}$. We start with its definition in Def. 2.

Definition 2 (Resource-Augmented PST Instance Model). *Let RA-PST be a resource-augmented PST constructed based on Alg. 1 for a process model P=(N,E) reflected by PST and a set of resource profiles RP. The resource-augmented PST instance model RA-PST$_{instance}$ is configured from RA-PST by selecting one resource allocation branch for each $n \in N$, applying all change operations contained in these branches to the PST, resulting in instance task set $N' = (N \cup N_{added}) \setminus N_{delete}$, and assigning to each task $n \in N'$ the resource specified as child node of the task (N_{added}: all tasks added by insert/replace operations; N_{delete}: all tasks removed by delete/replace operations).*

Consider the RA-PST$_{instance}$ model shown in Fig. 2c for a selection of Branch 1 for task `Evaluate Risk`, Branch 2 for task `Create Proposal`, and Branch 1 for task `Decide on Proposal`. The application of inserting `Confirm Risk` and the deletion of `Decide on Proposal` result in $N' =$ {`Evaluate Risk, Confirm Risk, Create Proposal`} constituting RA-PST$_{instance}$. The resource allocation results in: first the risk is evaluated by an associate, followed by confirming the risk by a trainee, and finally creating the proposal by a manager

Not every selection and combination of resource allocation branches in a RA-PST will result in an instance model that can be executed in the sequel: change patterns might not be applicable and/or tasks might exist for which no resource can be allocated. For the RA-PST in Fig. 2, for example, choosing Branch 2 for task `Decide on Proposal` will result in inserting a task for which no resource profile exists and hence no resource can be allocated. Hence, a validity notion for the RA-PST$_{instance}$ is required (see Def. 3).

Definition 3 (Valid Instance Model). *Let RA-PST be a resource-augmented PST constructed based on Alg. 1 for a process model P=(N,E) and a set of resource profiles RP. Then a RA-PST instance model RA-PST$_{instance}$ configured based on RA-PST is valid iff PST$_{instance}$ exists on $N' = (N \cup N_{added}) \setminus N_{delete}$ and $\forall n \in N' \; \exists$ a valid resource allocation $ra : N' \mapsto RP$ with $ra(n) = (rp.r, rp.role, rp.attr)$.*

In the following, we show how validity as defined in Def. 3 can be checked at the model level, i.e., on the RA-PST. Being able to check validity before process configuration contributes to efficiency when searching for a valid configuration.

Theorem 1 (Validity of Instance Model). *Let RA-PST be a resource-augmented PST constructed based on Alg. 1 for a process model P=(N,E) and a set of resource profiles RP. Then, a resource allocation represented by a branch in RA-PST with leaf node ln is valid if*

$$type(ln) = rp\_node \vee (type(ln) = ct\_node(Delete(x)) with x \in N) \quad (1)$$

with type() determining the node type.
A valid resource-augmented PST instance model RA-PST$_{instance}$ can be configured based on RA-PST by selecting one valid resource allocation branch $\forall n \in N$.

Proof. We first prove the validity conditions for a resource allocation branch in a RA-PST. Based on the construction of the RA-PST, we differentiate the following three cases, i.e., a resource allocation branch has leaf node of type i) resource profile, e.g., Branches 1–3 for task `Evaluate Risk`, ii) insert/replace, e.g., Branch 2 for task `Decide on Proposal`, and iii) delete, e.g., Branch 2 for task `Create Proposal` (all cf. Fig. 2a).
i) Branch leaf is a resource profile: Here we can differentiate two cases, i.e., the resource allocation branch i1) does not contain any change pattern nodes or i2) it contains insert/replace change patterns. Note that it is not possible that a resource profile node becomes a leaf node referencing a delete change pattern, i.e., if a delete change pattern node exists in the RA-PST, it is a leaf node. For case i1), validity is fulfilled for the allocation branch, as no change is applied to the PST and the resource of the resource profile leaf node can be assigned to the task, e.g., for Branch 2, allocating resource `Manager` to task `Evaluate Risk` is a valid solution. For case i2), the resource profile in the leaf node shows that all insert/replace patterns in the resource allocation branch can be resolved, i.e., they insert tasks for which, again, resource profiles and subsequently a resource to be allocated exist. For Branches 1 and 2 for task `Evaluate Risk` in Fig. 2a), for the insertion of task `Confirm Risk`, two resource profiles can be found; both are valid resource allocations.
ii) Branch leaf is insert/replace task: In this case, the resource allocation branch does not reflect a valid resource allocation as, obviously, for the newly inserted task no resource profile exists, hence no resource allocation is possible. In Fig. 2a, Branch 2 for task `Decide on Proposal` reflects an invalid branch (indicated by red rectangle) as for task `Supervise Decide on Proposal` to be newly inserted into the PST, no resource to be allocated exists.
iii) Branch leaf is delete task: If a branch ends with a delete pattern (e.g., Branch 2 for task `Create Proposal` in Fig. 2a), the resource allocation branch is valid if the task to be deleted exists in the process model, i.e., is in $x \in N$. In the example, `Decide on Proposal` exists in the PST, and hence, resource `Manager` can be allocated to task `Create Proposal` as in the RA-PST shown in Fig. 2b.

From i)–iii), we can conclude the validity pre-condition for single resource allocation branches in Theorem 1.

Now we prove that the RA-PST$_{instance}$ model that is configured based on valid resource allocation branches $\forall n \in N$ is valid, i.e., 1) the process model on

$N' = (N \cup N_{added}) \setminus N_{delete}$ forms a PST and 2) $\forall m \in N'$ a resource can be allocated.
1) N' forms a PST: N forms a PST (cf. Alg. 1). Insert, replace, and delete, in this work, only operate on tasks, i.e., per se sound SESE fragments [1]. Insertion does not violate block-structuredness as tasks are inserted directly before, after, or parallel to an existing task in the PST (cf. Def. 1). Replace internally deletes an existing task from the PST and inserts another task at the exact same position. Delete can be correctly applied if the task to be deleted exists [21]; this follows from (1) in Theorem 1.
2) $\forall m \in N'$ a resource can be allocated: As we select a valid resource allocation $\forall n \in N$, the tasks in $N \setminus N_{delete}$ are supplied with a resource. For the newly added tasks $\in N_{add}$, (1) in Theorem 1 ensures that the associated insert/replace operation is not a leaf node, i.e., it exists a resource profile for each of these tasks and hence the resources from these profiles can be allocated. □

While the RA-PST$_{instance}$ built from valid branches is valid, creating a configuration from invalid branches, can also result in a valid RA-PST$_{instance}$, i.e. if task `Supervise Decide on Proposal` in Fig. 2a would be deleted, Branch 2 would become valid. Such configurations might result in even better solutions in terms of the optimization goal (e.g. cumulative costs) and thus need to be considered by optimization approaches. The validity of the RA-PST$_{instance}$ must then be checked during the application of the change patterns at configuration.

To configure the RA-PST$_{instance}$, for a combination of branches for N, the change patterns contained in a branch are applied to build the RA-PST$_{instance}$. The change patterns *cp* are applied in a serial manner from task n to the corresponding leaf. In coherence with cases i) & ii), the insertion and replacement of tasks in serial manner is feasible, since no interdependency to other branches exists. In contrast, the implication of the delete pattern is not restricted to a local branch. The delete pattern can delete (invalid) parts of a different branch, making the order of applying the change patterns critical. We advocate to delay the application of a branch containing a delete pattern until all other branches are configured. In doing so, the number of valid solutions is maximized. Later inserted tasks can fulfill the pre-condition for delete.

Thus, to configure the RA-PST$_{instance}$ as shown in Figure 2c, for *Branch 1* of task `Evaluate Risk`, the task `Confirm Risk` is inserted. The application of change patterns for Branch 2 of `Create Proposal` are delayed, and Branch 1 for `Decide on Proposal` is allocated first. Then the delete pattern is applied on `Decide on Proposal`. As a result, `Decide on Proposal` is deleted.

Deleting a task requires also to roll back previously applied change patterns. As shown in Figure 2a) deleting the task `Decide on Proposal` does require the subsequent deletion of the task `Supervise Decide on Proposal` if it has been inserted earlier. A cascade delete of all children of the given task is done. Here the RA-PST structure is indispensable to resolve the delete operation.

[1] In future work, we will enable the insertion, replacement, and deletion of fragments.

Note, at the current state of implementation, we refrain from further ordering multiple delete operations. Deletes are applied in coherence with their position in the RA-PST. This might lead to a loss of a few valid solutions in cases where two delete operations target tasks in the same branch. Yet, the resulting solution is often the same. This loss could be mitigated by deleting tasks first, that are closest to leaves in the RA-PST. Suppose in the given example, a second delete pattern to delete task `Supervise Decide on Proposal` is present. If delete pattern `Decide on Proposal` is applied first, the second delete pattern `Supervise Decide on Proposal` would render invalid. Meanwhile, if `Supervise Decide on Proposal` is deleted first, also `Decide on Proposal` can be deleted successfully. The resulting RA-PST$_{instance}$ would be equal.

The RA-PST$_{instance}$ depicted in Fig. 2c reflects only one combination of resource allocation branches in the RA-PST. Figure 2b(top) depicts the set of all possible combinations of branches, i.e., 12, regardless of whether or not they are valid. We refer to any combination of branches as *solution* and the set of all solutions as *solution space*. The size of the solution space can be determined by multiplying the number of branches for each task $n \in N$. A solution is valid iff the resulting RA-PST$_{instance}$ is valid for the given configuration. The cost of a solution is the sum of costs of all allocations in the RA-PST$_{instance}$.

Determining a process configuration (solution) with a valid and a (near-) optimal allocation of resources for all tasks $n \in N$ in a process P is a **combinatorial optimization problem**. The traditional resource allocation problem is a general assignment problem [14]. In our case, the general assignment problem is extended by multiple aspects: First, a resource can be assigned to multiple tasks. Further, assigning a resource to a task might change the total number of needed assignments (cf. case iii)). As shown, standard solution methods for the assignment problem, such as a heuristic choosing the cheapest valid branch, might not find an optimal solution to this combinatorial problem, especially since the dynamic number of needed assignments increases complexity. Representing the solution space, consider Fig. 2b, where in the middle the valid solutions, and at the bottom, the unique valid solutions that remain after applying the change patterns are depicted.

4 Solving the Combinatorial Optimization Problem

Typical solution approaches for combinatorial optimization problems are simulated annealing, tabu search, and genetic algorithms [3]. When comparing the different solution approaches to problems of resource allocation and scheduling, [3] finds that the genetic algorithm performs best. For the presented problem, the interdependencies between allocations caused by change patterns are expected to further reduce the quality of simulated annealing, and tabu search [3]. Instead of these local search approaches, genetic algorithms identify good "building blocks" in different areas of the search space and combine them. They are then passed on to the next generation and recombined with other good building blocks to eventually find a good solution [10]. Another argument for using genetic algorithms

is their capability to solve problems with multiple optimization objectives, e.g., multiple resource attributes which will become important in future [11,4].

Figure 3 shows the different parts of a genetic algorithm adapted to the running example shown in Figure 2. The main challenge is to map the allocation problem by splitting the RA-PST into such a building block structure. Therefore, an encoding of the branch structure (cf. Figure 2) is applied. One individual describes a possible combination of different branches of the RA-PST and therefore represents one solution of the solution space. By considering invalid branches, we expect the genetic algorithm to find better solutions to the allocation problem than a heuristic considering only valid branches (cf. Theorem 1). In our case, one gene of the individual encodes the chosen branch for one task $n \in N$ for process P.

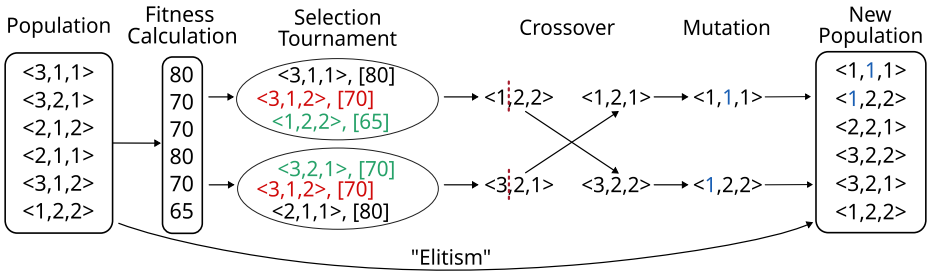

Fig. 3: Genetic algorithm with elitism.

An initial population is built from a random set of solutions out of all solutions in the solution space (cf. Fig. 2b(top)). These solutions are created by randomly assigning one allocation branch to each task $n \in N$ of process P.

A binary tournament selects the best individuals. In each tournament, a fixed number of individuals are randomly chosen from the population and compared by their fitness. The individual with the best fitness in the tournament is then picked for further evolution [1]. The tournament selection allows to deal with invalid solutions in a straightforward manner. The best valid solution in the tournament is picked as long as at least one solution in the tournament is valid. If no valid solution is available in the tournament, a random invalid solution is picked for the evolution. Therefore, no penalty calculation is needed for invalid solutions. The two individuals for evolution are found by holding two tournaments.

The fitness of one solution is defined as the sum of all costs of the encoded solution (cf. Fig. 2). Thus, in order to calculate a solution's fitness, the encoded individual must be decoded to the RA-PST$_{instance}$ as described in Sect. 3. The two individuals selected in the tournament are crossed with each other to create a new offspring solution. The position of the crossover point in the individual is chosen randomly. Following, the offspring is mutated to enable better exploration of the search space. These operations can be fully done on the encoded representation and do not need any decoding back to the RA-PST$_{instance}$.

In terms of improving run-time, [17] propose the early abandoning once no better solution can be found in a certain amount of iterations. Based on a testing phase, the authors decided to stop the evolution after twelve generations without improvement. Another improvement in terms of design and the evolution of the population is called elitism, which ensures that the best-fitting individual in a population is carried over into the next population without any changes. By doing so, the final population must contain the best individual found throughout all generations.

Experiments: To see if the genetic algorithm can find better solutions than the cheapest valid branch heuristic, both approaches are applied on the process model shown in Fig. 4. Different resource description files were generated for this process based on the description of resource profiles given in Sect. 2. Table 1 shows the metadata for the resource configurations used in the experiments. For each resource configuration, the RA-PST is built and given to the solution search algorithm.

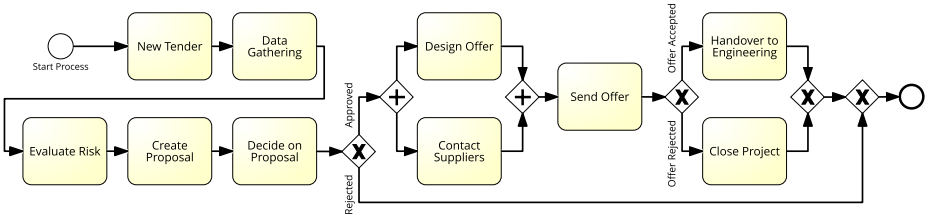

Fig. 4: Process model enhancing example in Fig. 1a (using SAP Signavio)

The configurations are designed by the authors to cover different scenarios, such as resulting in many invalid branches (`invalid_branches`), resulting in many configurations with similar costs (`close_maxima`), or having no deletes in the configuration (`no_deletes`). Two resource sets are created (in part) randomly. In set `heterogen`, the change patterns and resources are set manually, and the attribute costs for each resource profile are randomly assigned a value between 0 and 150. For the `fully_synthetic` set, the change patterns and resource profiles are added randomly for certain tasks. Comparable to [9], the configurations cover scenarios in which a genetic algorithm might struggle, i.e., a global optimum in a small region surrounded by a region of low fitness [1].

To evaluate the performance of the genetic algorithm, we compare the basic genetic algorithm, the same genetic algorithm with the notion of elitism, and the cheapest valid branch heuristic. The cheapest valid branch heuristic finds a solution by only choosing the best valid branch per task n (cf. Sect. 3, Theorem 1). Both genetic algorithms stop if, in twelve generations, no new best solution is found. As a quality benchmark, the optimum solution for each resource configuration is found by iteratively searching the full solution space. Due to the high computational complexity, searching the full solution space is not feasible for real-world applications. Since the genetic algorithm is non-deterministic, it is not sufficient to consider a single execution for the evaluation. We compare

	#Solutions	#Res.	#RPs	#CPs	Ins.	Del.	Rep.
heterogen	537600	5	49	11	7	4	0
no_deletes	691200	5	49	11	9	0	2
invalid_branches	1658880	6	51	20	15	5	0
close_maxima	4147200	6	55	14	10	4	0
fully_synthetic	7779240	5	49	36	24	8	4

Table 1: Overview of the different resource configurations used for the experiments sorted by the size of the resulting solution space. The columns show the number of resources (#Res.), resource profiles (#RPs), change patterns (#CPs) & the distribution over the change pattern types

the mean outcome of ten runs of the genetic algorithm with the heuristic and iterative approach. As metrics, the computation time and the best found solutions (for genetic algorithms, the average of the best in ten runs) are compared to the benchmark. To better evaluate the variance in outcomes of the genetic algorithm, the average delta to the optimum solution is given, as well as the standard deviation. The percentage of times the algorithm found the optimum out of the ten tries is given as optimum percentage.

Results: Table 2 presents the results of the experiments. Both genetic algorithms are run with the same parameters and have not been object to any parallelization or hyperparameter tuning. The focus of the experiments is to apply a genetic algorithm to the stated problem as a baseline for future development and to test if a genetic algorithm can find better solutions than the cheapest valid branch heuristic. The experiment data and results are available[2]. The results show that the genetic algorithms can find better solutions than the heuristic for the `close_maxima` and `invalid_branches` resource sets. Both approaches find the optimal solution in nearly all tries.

For the `fully_synthetic` resource set, the genetic algorithm with the elitist approach outperforms both the basic genetic algorithm and the heuristic approach. None of the approaches manages to find the optimal solution. The elitist approach gets the closest. On big solution spaces the genetic algorithm appears to find better solutions than the heuristic. As expected, the heuristic approach finds the optimum, if no deletes exist in the change patterns. The performance of the genetic algorithms for experiments with a small number of possible solutions is low. Only the elitist approach finds the optimal solution in some of the tries.

The results show that the adaptation of the genetic algorithm is feasible to solve the combinatorial resource allocation problem and can find better solutions than the cheapest valid branch heuristic. The comparatively bad performance of the genetic algorithm for resource sets with few branch interdependencies shows the genetic algorithms' weakness. A combination of the heuristic approach and the genetic approach could help to overcome this issue in the future. [11] proposes

[2] `https://github.com/Schlixmann/RA-RPST-Allocation`

Solver: / Dataset:		Benchmark	Heuristic	Plain Genetic	Elitist Genetic
heterogen size: 537600	time in sec	1661	0.01	17.97	6.54
	best	491.0	492.0	542.7	494.6
	delta (std)	0.0	**1.0**	51.7 (22.93)	3.6 (8.1)
	optimum perc.	100.0%	0.0%	0.0%	80.0%
no_deletes size: 691200	time in sec	1301.12	0.01	12.08	3.62
	best	492.0	492.0	575.9	501.1
	delta (std)	0.0	**0.0**	83.9 (29.33)	9.1 (12.33)
	optimum perc.	100.0%	100.0%	0.0%	50%
invalid_branches size: 1658880	time in sec	6198.89	0.01	26.47	5.54
	best	187.0	253.0	187.1	187.0
	delta (std)	0.0	66.0	0.1 (0.32)	**0.0 (0.0)**
	optimum perc.	100.0%	0.0%	90.0%	100.0%
close_maxima size: 4147200	time in sec	13904.79	0.01	8.14	3.72
	best	172.0	183.0	172.1	172.0
	delta (std)	0.0	11.0	0.1 (0.32)	**0.0 (0.0)**
	optimum perc.	100.0%	0.0%	90.0%	100.0%
fully_synthetic size: 7779240	time in sec	62618.45	0.01	34.87	10.5
	best	357.0	370.0	439.4	361.8
	delta (std)	0.0	13.0	82.4 (41.07)	**4.8 (5.79)**
	optimum perc.	100.0%	0.0%	0.0%	0.0%

Table 2: Comparison of different solution search approaches. Outcomes for the genetic algorithms as average over ten iterations. The table is sorted in ascending order based on the size of the solution space.

the initialization of the genetic algorithm with local optimal solutions instead of random ones. Applying this improved initialization could improve the genetic algorithm's performance and help to find an optimal solution quicker. By initializing with a good solution, the exploration features of the genetic algorithm can be exploited better. The RA-PST as a process modeling approach is independent of the used optimization technique. Other approaches than the genetic algorithm can be applied on the RA-PST and tuned to deliver good results.

5 Related Work

The proposed RA-PST approach is related to literature on process variability (see [22] for an overview). [2] provides work to enable variability on Configurable Refined Structure Trees (cPST). In contrast to the RA-PST approach, the configurable refined process structure trees define configurable areas as part of the tree by restriction and do not support multiple perspectives. Provop [13] proposes to design a base process model with variability points where new process fragments can be inserted. [12] shows typical issues with such bottom-up variability approaches. One is to find a suitable crossover point between the base model that should be extended and the separately modeled extensions. As most variable approaches, Provop focuses on the control-flow. An approach combining the variability of a process model with multiple process perspectives is shown in [15]. The authors combine multi-perspective i-EPCs with configurable objects and resources. The outcome is called C-iEPCs. Here, not only the control-flow is configurable, but also the resources and data objects that will be involved in the execution of the process. However, interdependencies between the different perspectives are not considered. The importance of this interconnection is pointed

out in [7], especially considering multiple dimensions. The RA-PST approach enables the analysis of such interdependencies and also uses them for performance optimization. [24] offers the possibility to automatically learn scheduling models from event data. The RA-PST could help improve the definition of these models and enable an easy transfer from BPM to scheduling optimization.

6 Conclusion

The presented core idea is that an allocated resource can actively change the control flow of a process instance. The RA-PST provides the formal framework for a process model that holds information on possible resource allocations and how they might affect the control-flow of the executed instance. It is also the basis for checking the validity of resource allocations and the configuration of RA-PST instance models. With the RA-PST a valid and good RA-PST$_{instance}$ can be found by a heuristic. The combinatorial optimization problem of determining a valid and optimal resource allocation is tackled with a genetic algorithm. We plan to optimize its performance further by, e.g., parallelizing the fitness calculation and through hyperparameter tuning. Besides genetic algorithms, the RA-PST provides a process structure that enables the application of advanced solution algorithms to the resource allocation problem. Hence, the RA-PST helps to further intertwine resource allocation and structural process requirements to achieve business process optimization.

Acknowledgements: This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - Project number 277991500

References

1. Beasley, D., Bull, D.R., Ralph, R.: An overview of genetic algorithms : Part 1, fundamentals. University Computing 15 (2) pp. 56–69 (1993)
2. Benallal, W., Barhamgi, M., Benslimane, D., Maamar, Z., Faci, N., Bellaaj, A.: A knowledge-based approach to manage configurable business processes. Concurr. Comput. Pract. Exp. **32**(15) (2020). https://doi.org/10.1002/CPE.4920
3. Connor, A.M., Shah, A.: Resource allocation using metaheuristic search. In: Computer Science & Information Technology. pp. 353–364 (2014)
4. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: Parallel Problem Solving from Nature. pp. 849–858 (2000)
5. van Der Aalst, W.M.P., La Rosa, M., Santoro, F.M.: Business process management: Don't forget to improve the process! Bus Inf Syst Eng **58**(1), 1–6 (2016)
6. Dijkman, R., Senderovich, A., van Jaarsveld, W.: 1st International Workshop on Data-Driven Business Process Optimization (BPO 2022) – preface. In: Business Process Management Workshops, p. 336 (2022)
7. Fahland, D.: Multi-dimensional process analysis. In: Business Process Management. pp. 27–33 (2022). https://doi.org/10.1007/978-3-031-16103-2_3
8. Fanjul-Peyro, L.: Models and an exact method for the Unrelated Parallel Machine scheduling problem with setups and resources. Expert Systems with Applications: X **5**, 100022 (2020). https://doi.org/10.1016/j.eswax.2020.100022

9. Ficarella, E., Lamberti, L., Degertekin, S.O.: Comparison of three novel hybrid metaheuristic algorithms for structural optimization problems. Comput. Struct. **244**, 106395 (2021). https://doi.org/10.1016/j.compstruc.2020.106395
10. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., 1st edn. (1989)
11. Guariso, G., Sangiorgio, M.: Improving the performance of multiobjective genetic algorithms: An elitism-based approach. Information **11**(12), 587 (2020)
12. Hallerbach, A., Bauer, T., Reichert, M.: Issues in modeling process variants with provop. In: Business Process Management Workshops, vol. 17, pp. 56–67 (2009). https://doi.org/10.1007/978-3-642-00328-8_6
13. Hallerbach, A., Bauer, T., Reichert, M.: Capturing variability in business process models: the provop approach. Journal of Softw. Maintenance and Evolution: Research and Practice **22**(6-7), 519–546 (2010). https://doi.org/10.1002/smr.491
14. Hillier, F., Lieberman, G.: Introduction to Operations Research. McGraw-Hill Education (2021)
15. La Rosa, M., Dumas, M., ter Hofstede, A.H., Mendling, J.: Configurable multiperspective business process models. Information Systems **36**(2), 313–340 (2011)
16. Middelhuis, J., Bianco, R.L., Scherzer, E., Bukhsh, Z.A., Adan, I.J.B.F., Dijkman, R.M.: Learning policies for resource allocation in business processes (2024). https://doi.org/10.48550/arXiv.2304.09970
17. Oliveto, P.S., Witt, C.: On the runtime analysis of the simple genetic algorithm. Theoretical Computer Science **545**, 2–19 (2014). https://doi.org/10.1016/j.tcs.2013.06.015
18. Peters, S.P.F., Dijkman, R.M., Grefen, P.W.P.J.: Resource optimization in business processes. In: Enterprise Distributed Object Computing Conference (EDOC). pp. 104–113 (2021). https://doi.org/10.1109/EDOC52215.2021.00021
19. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies (2012). https://doi.org/10.1007/978-3-642-30409-5
20. Rinderle, S., Reichert, M.: A formal framework for adaptive access control models. J. Data Semant. **9**, 82–112 (2007). https://doi.org/10.1007/978-3-540-74987-5_3
21. Rinderle-Ma, S., Reichert, M., Weber, B.: On the formal semantics of change patterns in process-aware information systems. In: Conceptual Modeling. pp. 279–293 (2008). https://doi.org/10.1007/978-3-540-87877-3_21
22. Rosa, M.L., Aalst, W.M.V.D., Dumas, M., Milani, F.P.: Business process variability modeling: A survey. ACM Computing Surveys (CSUR) **50**(1), 1–45 (2017). https://doi.org/10.1145/3041957
23. Schumann, F., Rinderle-Ma, S.: Resource-driven process manipulation: Modeling concepts and valid allocations. In: Cooperative Information Systems. pp. 416–426. https://doi.org/10.1007/978-3-031-46846-9_23
24. Senderovich, A., Booth, K.E.C., Beck, J.C.: Learning scheduling models from event data. In: Proceedings of the International Conference on Automated Planning and Scheduling. vol. 29, pp. 401–409 (2019). https://doi.org/10.1609/icaps.v29i1.3504
25. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and more focused control-flow analysis for business process models through SESE decomposition. In: Service-Oriented Computing. pp. 43–55 (2007). https://doi.org/10.1007/978-3-540-74974-5_4
26. Varvoutas, K., Kougka, G., Gounaris, A.: Optimizing business processes through parallel task execution. In: International Conference on Management of Digital EcoSystems. pp. 24–31 (2022). https://doi.org/10.1145/3508397.3564842