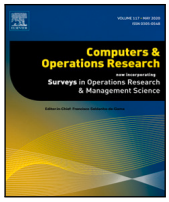
Computers & Operations Research 161 (2024) 106446

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor



Combining an LNS-based approach and organizational mining for the Resource Replacement Problem

Claudia Diamantini, Ornella Pisacane, Domenico Potena *, Emanuele Storti

Dipartimento di Ingegneria dell'Informazione, Università Politecnica delle Marche, 60121 Ancona, Italy

ARTICLE INFO

Keywords:
Process planning
Business continuity
Process mining
Integer Linear Programming
Matheuristic

ABSTRACT

For companies, it is crucial to promptly react to (even short-term) lack of resources, for guaranteeing the continuity of the operations in business processes. This leads to the solution of a *Resource Replacement Problem* (RRP) aimed at reassigning as many activities performed by resources that are no longer available to those that are available. To this purpose, several aspects are considered simultaneously, e.g., resources skills, workloads and other domain-specific constraints. In this paper, we propose an innovative hybrid approach for solving RRP, combining mathematical optimization with organizational mining. In particular, logs of past process executions are used to model a social network of resources by organizational mining techniques. Then, a similarity measure among resources is derived and exploited along with run-time resource workload and information on activities priority to formulate an Integer Linear Programming (ILP) model for reassigning the activities of unavailable resources, minimizing the total reassignment cost. To efficiently solve RRP, a Large Neighborhood Search based matheuristic is developed. Computational experiments show that the proposed matheuristic outperforms the commercial solver used to solve the ILP model. A sensitivity analysis, on possible variations of the input parameters and on the moves of the matheuristic, concludes the work.

1. Introduction

In organization management, resource replacement planning consists in addressing short-term lack of workforce that may be due to temporary or permanent unavailability of human (e.g., due to resignation, illness) or technical (e.g., due to hardware failure) resources. In a sense, the ability of a company to deal with the replacement of resources that are no longer available, for carrying out certain activities, becomes crucial to both ensure resilience and guarantee continuity of operations in business processes. Several different aspects have to be considered in the decision process like, for example, resource skills, workloads as well as possible other domain-specific constraints. In particular cases, such as emergency situations like a pandemic, the number of such unavailable resources to replace could be significant. As a consequence, the lack of a business continuity plan could lead to the interruption of the entire system as the replacement of unavailable resources becomes crucial especially for carrying out those activities that could represent bottlenecks for the completion of others.

Traditional approaches in the literature for resource scheduling and replacement planning address the problem typically through business rules modeled by domain experts. However, nowadays information systems are suitable to track and monitor every event occurring during a business process, such as what activities have been executed, when and by whom. Therefore, it is possible performing more advanced kinds of data-driven analysis, e.g., through process mining techniques.

This paper proposes a data-driven optimization-based approach for addressing the problem of replacing resources, namely, the Resource Replacement Problem (RRP). In the following, it is assumed to have a set of resources, a subset of which has to be replaced since they are temporary or permanently unavailable and a set of activities assigned to the unavailable resources that have to be reassigned. Each resource is characterized by some skills (e.g., it can perform all or a subset of activities), a current workload (computed considering the workload required by the activities it already manages) and a maximum workload (i.e., the resource capacity). Each activity requires some skills to be performed, takes up the capacity of the resource which is assigned to and has a priority. These priorities depend on both the causal relationships defined in the business process and other business rules (e.g., a process instance related to a given order has to be executed before another). Therefore, an activity cannot be executed before that all the higher priority activities have been already performed. The decisions taken by solving an RRP consist in reassigning the activities of the unavailable resources to the available ones under both priority

* Corresponding author.
E-mail addresses: c.diamantini@univpm.it (C. Diamantini), o.pisacane@univpm.it (O. Pisacane), d.potena@univpm.it (D. Potena), e.storti@univpm.it (E. Storti).

https://doi.org/10.1016/j.cor.2023.106446
Received 1 October 2022; Received in revised form 3 October 2023; Accepted 3 October 2023
Available online 12 October 2023

and workload constraints by minimizing the total reassignment cost. Such a total cost takes into account both the reassignment costs and the penalties due to those activities that remain unassigned. In particular, the reassignment costs are determined as a measure of both the similarity between two resources, such that similar resources are preferred, and workload balancing. In this paper we evaluate similarity with respect to several aspects, namely how much the two resources collaborate together, and the relative performance and experience on the activity to be performed. Such information are derived by analyzing past executions of business processes, and by modeling a sociogram, i.e., a graph of social relations among resources in terms of handover of work. An edge between resources A and B exists in the sociogram if, within a process instance, after A has performed an activity the work passes to B for the next activity.

On the basis of this information, the RRP is then modeled as an Integer Linear Program (ILP) and a Large Neighborhood Search matheuristic is also designed for efficiently solving it on medium-sized and large-sized instances.

The contribution of the paper is multi-fold. For what concerns the representation of social relations:

- we propose a novel definition of the sociogram modeling the handover of work relation. Unlike existing models in the literature, we explicitly represent the specific type of the transferred activity, enabling a greater expressiveness. Furthermore, a novel handover of work metric is introduced and compared with the others in the literature;
- we define, on top of the sociogram, a measure of similarity among resources by considering the degree of collaboration between two resources, the performance and the experience of a resource to perform a given activity.

Regarding the solution of the resulting RRP:

- we model the problem through Integer Linear Programming (ILP), with the aim of deciding how to reassign the activities performed by resources no longer available to those resources that are available, that have the skills and that have a residual workload sufficient to execute them. The objective function to minimize takes into account the cost of reassigning those activities plus possible penalties due to those activities that remain unassigned;
- we design a Large Neighborhood Search (LNS) based matheuristic in order to efficiently address also medium-sized and large-sized instances of RRP. To the best of our knowledge, this is the first work in which the proposed matheuristic is designed for solving an RRP;
- we perform a sensitivity analysis on possible variations of some input parameters and on the moves implemented in the matheuristic. We believe that the scientific community can take advantage from these considerations for future implementations of such an approach applied to similar decision problems.

The rest of this work is structured as follows: Section 2 reviews related work, whereas Section 3 presents a running example that is used throughout the paper. Section 4 introduces the preliminary terminology, a novel handover of work metric and the sociogram model. Section 5 discusses the overall methodology for RRP. Section 6 describes the ILP model proposed in order to select the best resource replacement whereas Section 7 details the LNS-based matheuristic developed in order to address efficiently large-sized instances of the problem. Numerical comparisons between the ILP model and the matheuristic are discussed in Section 8. Finally, Section 9 concludes the work and draws some future research directions worth of investigation.

2. Related work

Resource replacement

The interest towards semi-automatic solutions supporting the assignment of resources to activities in organizations is witnessed by the number of approaches that have been developed in recent years. Allocating resources is often seen as an optimization problem. Indeed, a variety of approaches have been comprehensively discussed in several surveys on the topic (De Bruecker et al., 2015; Pufahl et al., 2021), differing with respect to (a) the criterion/a to be optimized, (b) the role of process model and the related information used to drive the decision, and (c) the adopted solution technique. The majority of work is reportedly focusing on *process-oriented* optimization, aimed to find the resources that best fit the activity to replace, with a one (activity)-to-one (resource) approach or a more complex one-to-many or many-to-many allocation. This can be done either with a local or a global perspective, in which different priorities of running process instances (and related activities to perform) are considered. Other possible measures to optimize include the process cost, the cycle time of the process or the throughput. A different category of *resource-oriented* approaches are aimed to balance workload, thus assigning an activity not necessarily to the best-fitting resource, or to optimize the worklist of to-do activities for each resource.

Depending on how the problem is modeled, different categories of approaches can be devised. The simplest of them is based on *rules*, e.g., constraints or manually-defined guidelines, that frequently allows to find a feasible solution in less time than alternative approaches, although with fewer guarantees on its quality. For example, Kumar et al. (2002) argue that in the scenario of workflow processes there is a dynamic trade-off between quality and performances, which should be considered when assigning activities to resources (e.g., considering approaching deadlines may enable to offer work to less qualified workers). To this aim, such a work proposes a comprehensive mechanism for activity allocation by defining a metric including the notions of suitability, urgency, conformance and availability. However, such a metric is manually defined from domain knowledge, whereas in our approach metrics are derived from logs in a data-driven fashion. More advanced logic programming rules are also proposed, for instance in the form of Answer Set programming, to formally model dependencies and conflicts across resources used to derive a feasible schedule (Havur et al., 2016).

Among the works following a data-based *inductive approach*, several data mining techniques are explored to derive rules for resource assignment, such as declarative mining, reinforcement learning, decision trees, association rules, or support vector machines (Liu et al., 2008). Instead, other works aim at supporting the allocation of activities through modeling of social relations by Hidden Markov Models (Yang et al., 2008), for instance; others include genetic algorithms or meta-heuristics.

On the other hand, several literature contributions make use of mathematical programming to deal with the problem of assigning resources to activities. Mathematical models are formulated to minimize cycle time. For example, considering input/output of activities and precedence, Hirsch and Ortiz-Peña (2017) formulate a Mixed Integer Non-Linear Program and design a set of heuristics. It is worth noting that they address a different problem than the one proposed in the present work. In particular, they model the process operating phase within the mathematical formulation and then, they directly consider the inputs and the outputs of each activity. It means that, in their model, constraints ruling the collaboration mechanism among resources are also imposed. Moreover, unlike our work, they plan and schedule the activities without assuming that some resources may be unavailable. It means that they do not have resources with an already fixed workload (i.e., resources that already have to perform a certain set of activities). From the mathematical model perspective, unlike

us, they assume that all the activities have to be assigned and their objective (to minimize) is the maximum completion time. Arias et al. (2018) propose a flexible system considering multiple criteria, such as information from past executions of the process (e.g. frequency, performance, quality), the required skills to perform each activity and their resource workload. Then, an ILP model is formulated to allocate a single resource to an activity and a heuristic is used for batch resource recommendation. However, unlike our work, they do not address a reassignment problem. In fact, they do not take into account the priorities among the activities as well as the similarity among the resources. Finally, the proposed heuristic (named BPA2) is used for ranking the resources for a specific activity in order to obtain a prioritized list of the most suitable resources on an individual basis.

Xie et al. (2016) develop a dynamic task assignment approach for minimizing the cycle time of business processes at the run-time stage. They refer to an individual worklist model, where each resource has a predefined list of assigned activities to perform. By relying on both stochastic and queuing theory, for each resource, the system schedules the activities to execute from the corresponding list. On the other hand, a list of shared tasks can be dynamically assigned to idle resources by role type. The problem they address is different from ours, as they minimize the cycle time, by taking into account only the arrival rate of the run-time requests for the tasks.

Finally, *heuristic* approaches are extensively used for their capability to balance solution quality and computational effort. As shown in Pufahl et al. (2021), these solutions include a multitude of different approaches. Among them, Cabanillas et al. (2013) propose a semantic-based methodology in which the allocation of activities to resources is performed considering the ranking of resources according to user-defined preferences. However, the ranking mechanism does not apply any mathematical optimization techniques. Instead, other contributions are focused on minimizing cost or cycle-time of the process (e.g., Zhao et al., 2017 which apply Particle Swarm Optimization or Huang et al., 2012 which exploit Ant Colony Optimization).

Therefore, to the best of our knowledge, in the literature, there are no works dealing with a reassignment problem like ours. It is worth noting that, in a reassignment problem, a list of activities already assigned to resources must exist. Moreover, in the literature, the goal of an assignment problem is almost always to minimize cycle time. Instead, in the RRP, the goal is to maintain at least the operations efficiency defined in the assignment phase.

Resource profile modeling

Properly modeling resources is a necessary step for analyzing processes from an organizational perspective in most of the replacement approaches. Information on resources can be retrieved from either models/process schemas (in a top-down approach), or data produced by process execution (according to a bottom-up approach), although a mix of the two approaches is frequently used.

An approach that is gaining increasingly popularity consists in deriving information on resource capabilities and social relations from logs of past process executions, through *organizational mining* techniques (Song and van der Aalst, 2008). The task of discovering a model from an event log means in this case to build a model of social relations among organizational entities. Such a model may be aimed to represent the current organizational structure or a social network of communication or interaction in the organization (Van Der Aalst et al., 2005). This can be analyzed through common Social Network Analysis techniques (Wasserman and Faust, 1994). In this context, several metrics have been devised to analyze different relations among resources, including handover of work (transferring work between two resources), subcontracting (a resource executed an activity in-between two activities executed by another resource), the reassignment of an activity to a resource and the cooperation of different resources on an activity. In particular, Van Der Aalst et al. (2005) discuss several variants for each of the aforementioned metrics, defined according to which specific aspects are considered (e.g., self-loops, causality) and present a plugin for ProM (Van Dongen et al., 2005) implementing the metrics. An extended set of metrics are proposed in Pika et al. (2017), including aspects such as skills, productivity, utilization, and collaboration patterns from event logs. Most of the organization mining approaches focus on discovering groups of similar resources, by analyzing logs to extract, for each of them, information on the performed activities and then, modeling it through a matrix representation. Although a simple yet effective way to represent information, the limitation to such two dimensions may mean not taking context into account (e.g., under what circumstances an activity has been performed).

Some exception to the mentioned approaches investigates alternative methods. For instance, an approach based on context-aware resource profiling is proposed in van Hulzen et al. (2021), also considering case attributes and variables capturing the system state. The method allows activities belonging to multiple profiles simultaneously, discerning specialists from generalists.

On the other hand, in Appice (2018), a modified Louvain algorithm is adopted, with the purpose to discover overlapping organization units instead of the traditional discovery of disjoint resources. By considering a windowed stream of events instead of a static event log, the approach supports the analysis of the life cycle of the dynamic organization of a business process.

Although with a different goal, namely enhancing a process from the organizational perspective with information on roles, in Burattin et al. (2013), social relations among resources are analyzed from event logs. Given a process model and a log in input, the approach aims to partition the set of activities of the process by roles. This partitioning is performed by grouping originators (i.e., the resource transferring work to another resource) in roles and associating activities with the corresponding role.

In Lee et al. (2019), a metric is defined through process mining and social network analysis, called Degree of Substitution, that is used to quantify the overlap in the work experiences of human resources. The goal is to select the most suitable replacement for a given unavailable resource and activity to perform. The work makes an assumption similar to ours, namely that a resource can be most effectively replaced by another that either works on similar activities or has similar transfers of work. In Schönig et al. (2016), an organizational mining framework is proposed for the discovery of patterns related to resource assignment using declarative process modeling languages based on rule templates. Unlike our work, the approach is used only for describing, instead of enacting, the resource replacement. Measures derived from joint activities, capable to determine a profile based on how frequently a resource performs specific activities, have also been investigated. For instance, in Van Der Aalst et al. (2005), a joint activities matrix is obtained with resources on rows and activities on columns. Various distance metrics are proposed, among which the Minkowski distance, the Hamming distance and the Pearson's coefficient. While the Minkowski distance is only suitable when resources perform a comparable amount of work, the others are more robust, with some adjustments when the volume of work varies significantly (e.g., part-time versus full-time workers). An approach similar to ours is followed by Conforti et al. (2015), although the work has a different purpose, namely proposing a recommendation system to support process participants to take a risk-informed decision in choosing the next activity to execute out of a set of assigned activities.

3. Running example

The example proposed in this section is related to the process to repair telephones in a company and will be used throughout the paper. Hereby, the process schema is represented as a Petri net (Peterson, 1977). In Petri nets, a *transition* represents a process activity, i.e., a task

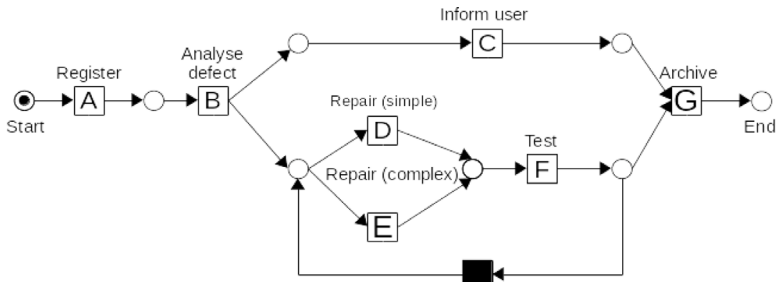


Fig. 1. The example process model, represented as a Petri net (transitions are shown as boxes and places as circles).

Table 1
Running example: event log for the repair process. Time unit is minute.

ID	Trace
1	(A, James, 15), (B, Mark, 60), (C, James, 8), (D, Harrison, 25), (F, Alec, 18), (G, Carrie, 9)
2	(A, Carrie, 12), (B, Harrison, 74), (E, Harrison, 79), (F, Peter, 17), (C, James, 4), (G, James, 13)
3	(A, Carrie, 10), (B, Peter, 43), (D, Peter, 43), (F, Alec, 72), (C, Carrie, 7), (E, Harrison, 115), (F, Alec, 20), (G, James, 14)
4	(A, James, 20), (B, Mark, 96), (C, James, 3), (D, Mark, 31), (F, Alec, 209), (D, Alec, 53), (F, Peter, 23), (G, Carrie, 11)

that has to be performed within the process and that is interconnected to other transitions through *places* which may contain zero or more tokens. Transitions with multiples ingoing or outgoing places represent AND gateways, whereas places with multiples ingoing or outgoing arcs stand for XOR gateways. A transition is *enabled* if each input place, i.e. a place that is linked in input to the transition, contains at least one token. In this case, the transition can fire (i.e., the corresponding activity can be performed). As a result, a token is consumed from each input place and a token is placed in each output place. As such, the distribution of tokens in the places of the Petri net defines what transitions are enabled and therefore, describes the possible evolution of the process. In Fig. 1, the Petri net for the example is shown, where transitions are depicted through boxes, and places through circles. A single token is placed in the Start place. The black box represents a hidden transition which is used only for routing purposes.

At the beginning, the only activity that can be executed is *Register*, as the only place with a token is the Start place. After the registration of a repair request (A) is done, an analysis of defects is performed (B). Then, the user is informed about the outcome (C) and the repair subprocess is performed in parallel (in fact, after B each output place contains a token). This is achieved by executing either a simple repair (D) or a complex one (E), and is ended by a test to verify whether the repair solved the issues (F). In the negative case, the repair subprocess is re-executed (i.e., the token is moved before D and E). Finally, if the test succeeds, the request is archived (G) and the process ends when the token is in the End place. Six users are involved in the process. In particular, James and Carrie can be assigned to the administrative activities (A, C, G) whereas, Mark, Alec, Harrison and Peter to the analysis, repair and test activities (B, D, E, F).

Each record in the log refers to a specific process instance (e.g., same repairing process performed for different requests) and is a sequence of events, each of which described in terms of performed activity, resource and duration in a certain time unit, as shown in Table 1. It is worth remarking that the control flows in the process model in Fig. 1 are flattened in the log. For example, the XOR gateway between the two repair activities (D and E) does not appear in the log because only one of them is actually executed.

4. Representation model for resources and activities

In this section we introduce the terminology and a model of social network for handover of work that is used in the rest of the paper.

4.1. Activities and resources

Hereby we refer to the term *activity* as a task (or portion thereof) performed to achieve a goal, and to *resource* as any organizational entity that is capable to perform some activities, including not only personnel but more in general machines, software, agents.

Definition 1 (*Event, Trace & Event Log*)**.** Let $\mathcal{A}$ be a set of activities and $\mathcal{R}$ a set of resources. $V = \mathcal{A} \times \mathcal{R}$ is the set of possible *event*s, i.e., combinations of an activity and a resource. Given a resource r, $V(r) \subseteq V$ is the set of events in which r can take part. A *trace* is a possible sequence of events, where $C = V^*$ is the set of all possible traces. An event log $\mathcal{L}$ is a subset of all bags (multi-sets) over C (Van Der Aalst et al., 2005).

In real contexts, each event in an event log can be characterized by a set of additional attributes. In our scenario, we consider at least the case id to identify the process instance, the executed activity, the name of the resource, the time needed for its execution. For convenience, we introduce the function $\pi_a(\sigma)$ which returns the set of activities related to events of a trace σ. In the following, we provide a detailed characterization of activities and resources, in terms of average load request for an activity, resource skills and load. All the measures provided below are given with respect to a *reference period*, e.g., a working day or week, that can be defined arbitrarily at the application level.

Definition 2 (*Average Load Request For Activities In A Reference Period*)**.** Given an event log $\mathcal{L}$, an activity $a_i \in \mathcal{A}$, a reference period $p = (t_0, t_1)$ where t_0 and t_1 are the starting and the ending timestamps respectively, the average load request $\lambda(a_i, \mathcal{L}, p) \in [0, 1]$ for a_i in the reference period p is computed as:

$$\lambda(a_i, \mathcal{L}, p) = \frac{\sum_{\sigma \in \mathcal{L}_p} execution\_time(a_i, \sigma)}{\sum_{\sigma \in \mathcal{L}_p} num\_occurrences(a_i, \sigma)} \cdot \frac{1}{length(p)}$$

Here, $\mathcal{L}_p \subseteq \mathcal{L}$ is the subset of traces executed within the given reference period p, $execution\_time(a_i, \sigma)$ is a function returning the total execution time for all the occurrences of activity a_i for a specific trace σ, while the function $num\_occurrences(a_i, \sigma)$ returns how many times the activity a_i occurred in σ. Finally, the ratio is normalized by the length of the reference period, computed as $length(p) = t_1 - t_0$. The specific unit of measurement to use for time and for the length of the reference period (e.g., second, minutes) is left to implementation. The only requirement is that the unit for the reference period and the execution time must be the same (or must be scaled accordingly). In the following, whenever not ambiguous, we refer to λ_i omitting the other parameters for clarity.

By referring to the running example of Section 3, let us consider activity D and a reference period of 8 h, i.e., 480 min. The activity occurs 4 times, with execution times equal to 25 (case 1), 43 (case 3), 31 and 53 (case 4) min. The total execution time is therefore 152 min, and the average execution time is 38 min. Finally, λ for activity D is equal to $\frac{152}{4} \times \frac{1}{480} = 0.08$. The evaluation of the execution time of an occurrence of an activity depends on the information recorded in the

event log. If both the starting and ending event have been recorded, the execution time is given by the difference of the corresponding timestamps. On the other hand, if only the starting event is recorded, it can be obtained from the information on its duration, if available. If this last is not reported, then execution time can be estimated by the difference between the timestamp of the causally following event and the event at hand (for details, we refer the reader to the discussion reported below on the handover of work).

Resources can be characterized in terms of their skills (namely, the activities that they performed in the past) and their workload. This last is measured both as the current and maximum workload, as defined below.

Definition 3 (*Skills Of Resources*)**.** Given an event log $\mathcal{L}$ and a resource $r_i \in \mathcal{R}$, the skills of r_i is the set $\mathcal{A}(r_i) \subseteq \mathcal{A}$ of activities that r_i performed at least once in a trace of $\mathcal{L}$.

To make an example, from the event log in Table 1 we derive $\mathcal{A}$(Mark) = {B, D} and $\mathcal{A}$(Harrison) = {B, D, E}.

Definition 4 (*Workload Factors Of A Resource*)**.** Given a resource $r_i \in \mathcal{R}$, $\mu_i \in [0, 1]$ is the maximum workload for r_i, while $\gamma_i \in [0, 1]$ is the current workload factor of r_i, with $\gamma_i < \mu_i$.

While the latter refers to the current workload experienced by a resource, the former typically refers to a certain reference period. For instance, γ_{James}=0.5 means that the resource James is currently experiencing a workload of half its capacity (in the literature, sometimes this notion is named resource availability, e.g., in Martin et al., 2020). Usually, for a resource r_i we can assume that μ_i is equal to 1, meaning that it can be assigned to activities for the full working time, e.g., the reference time may be set to 8 h for a working day. Given the possible wide variety of policies on working times in different organizations, the specific computation formula for γ is left to implementation. In this work, we estimate the current workload for a resource considering the working time within the time reference period at hand (i.e. the sum of the load time for all the activities already performed in the period), as well as the expected time to complete possible on-going activities. Other data-driven approaches are proposed in the literature and may be adapted for the present scenario. For example, in Martin et al. (2020), the authors consider also concurrent activities and intermediate interruptions (e.g., due to a break) in the definition of an availability calendar.

4.2. Handover of work

Among the various social relations that can be recognized in an event log between two resources, we focus here on relations of possible causality, and specifically on *handover of work*. Within a case, there is a handover of work from a resource r_1 to a resource r_2 if there are two subsequent activities a_1 and a_2 where a_1 is completed by r_1 and a_2 by r_2. The metric can be defined and computed in multiple ways (Van Der Aalst et al., 2005), according to:

- the degree of causality, i.e., the length of the handover. This can either be direct (the second activity directly follows the first in the trace) or indirect if the length is larger than 1;
- the existence of multiple successions from a resource to itself (multiple self-transfer);
- the kind of succession: it is possible to consider either arbitrary transfer of work between two subsequent events in a trace or casual dependency between two activities in a trace. A process typically includes some parallel activities, but all events are recorded in a trace as a sequence. Hence, in order to identify parallel events within a trace, one needs to know which are the dependencies, or causal relations, existing between the activities of the process. A process model is needed in this case.

We provide in the following the definition of causal relation, from which we define handover of work.

Definition 5 (*Causal Relation*)**.** A Causal Relation (CR) is a relation on the set of activities, i.e., $CR \subseteq \mathcal{A} \times \mathcal{A}$. Given $a_1, a_2 \in \mathcal{A}$, $a_1 \rightarrow a_2$ denotes that $(a1, a2) \in CR$

Elements of CR represent the order of execution for a pair of activities and is derived from a process model. X → Y states that Y cannot be executed until X is terminated (the execution of Y depends on the execution of X). With reference to the case 3 in the example of Section 3, between events (A,Carrie) and (B,Peter) there is a direct and causal succession, while between (F,Alec) and (C,Carrie) there is a direct but arbitrary succession. Indeed, if we compare the log to the process model in Fig. 1, there is a causal relation A→B, but this does not hold between activities F and C.

In this work, we refer to the handover of work relation (1) by ignoring self-transfers, (2) by considering indirect succession and (3) causal relation, i.e., we take into consideration succession between activities, with any length, only if aligned to the process model. The reason behind (1) is that the research question of the work is related to resource replacement, for which considering handover from the same resource is not helpful. The reason of (2) and (3) is related to the need to correctly identify handover of work in real processes and avoid spurious relations. To detect causal relations, we rely on the approach discussed in Diamantini et al. (2016), which enables to make causal relations between events in a trace explicit. This is done by translating a trace in an instance graph (van Dongen and van der Aalst, 2004), where nodes represent events of the trace, while each edge represents a direct succession between two events, such that a causal relation among the corresponding activities holds. It is worth noting that a process model, if not available, can be discovered through process discovery algorithms, e.g., alpha miner, heuristic miner, inductive miner.

In the following, we define the function and the formula for computing the metric.

Definition 6 (*Handover of Work*)**.** Given $r_1, r_2 \in \mathcal{R}$, $a_x \in \mathcal{A}(r_1), a_y \in \mathcal{A}(r_2)$, an event log $\mathcal{L}$, a process model $\mathcal{P}$, a set of Causal Relations CR derived from $\mathcal{P}$, and a trace $\sigma \in \mathcal{L}$, the function $r_1 \otimes^{\sigma}_{a_x,a_y} r_2$ returns how many times r_1 and r_2 executed respectively activity $a_x \in \pi_a(\sigma)$ and $a_y \in \pi_a(\sigma)$ such that $(a_x, a_y) \in CR$. For the whole event log $\mathcal{L}$, the function is computed as:

$$q_{xy} = r_1 \otimes^{\mathcal{L}}_{a_x,a_y} r_2 = \sum_{\sigma \in \mathcal{L}} r_1 \otimes^{\sigma}_{a_x,a_y} r_2$$

Finally, the metric *handover of work* between r_1, r_2 for activities a_x, a_y in $\mathcal{L}$ is computed as:

$$h_{xy} = r_1 \odot^{\mathcal{L}}_{a_x,a_y} r_2 = r_1 \otimes^{\mathcal{L}}_{a_x,a_y} r_2 / \Big(\sum_{r_i \in R} \sum_{r_j \in R} r_i \otimes^{\mathcal{L}}_{a_x,a_y} r_j \Big)$$

The metric is computed for a pair of resources r_1, r_2 and with respect to a pair of activities a_x, a_y by dividing the total number of proper causal succession (with no self-transfer) by the total number of causal succession of a_1, a_2 between any two resources r_i, r_j, with $i \neq j$. In other terms, the metric evaluates how peculiar the relation between two resources in the execution of such two activities is. It is worth considering that by dividing for the total number of causal successions, infrequent relations will be taken into account too. For instance, considering the example in Section 3, given the resources James and Carrie, and the activities C and G, $James \odot^{\mathcal{L}}_{C,G} Carrie = \frac{2}{4} = 0.5$, because the causal succession between them happens 2 times (case 1 and case 4), but the causal succession of C and G between any two resources occurs 4 times.

This information can be represented as a *handover matrix*, that is a matrix $N \times N$, where $N \leq |\mathcal{R}|$ is the number of active resources (i.e., resources that executed at least 1 activity in the event log). Given two resources $r_i, r_j \in \mathcal{R}$ active in $\mathcal{L}$, the cell (i, j) of the handover matrix

Table 2
Running example: handover matrix.

	James	Marc	Carrie	Harrison	Alec	Peter
James		(A, B, 0.5, 2)	(C, G, 0.5, 2)			
Mark	(B, C, 0.5, 2)			(B, D, 0.3, 1)	(D, F, 0.25, 1)	
Carrie	(C, G, 0.25, 1)			(A, B, 0.25, 1)		(A, B, 0.25, 1)
Harrison	(B, C, 0.25, 1)				(D, F, 0.25, 1) (E, F, 0.5, 1)	(E, F, 0.5, 1)
Alec	(F, G, 0.25, 1)		(F, G, 0.25, 1)	(F, E, 1.0, 1)		(D, F, 0.25, 1)
Peter	(F, G, 0.25, 1)		(B, C, 0.25, 1) (F, G, 0.25, 1)		(D, F, 0.25, 1)	

includes a tuple $(a_x, a_y, h_{xy}, q_{xy})$, with $a_x \in \mathcal{A}(r_i)$ and $a_y \in \mathcal{A}(r_j)$, if and only if $h_{xy} > 0$. In Table 2, we report the handover matrix for the running example.

From the matrix, a *sociogram* can be defined, i.e., a graph of social relations where nodes represent resources and there exists an edge linking two nodes if a certain social relation is recognized between them. We refer to the following definition of a sociogram as a labeled multidigraph, i.e., a directed multigraph where two nodes may be linked by multiple labeled edges.

Definition 7 (*Sociogram*)**.** Given a $N \times N$ handover matrix M, a sociogram G is defined as an 8-tuple $G = (\Sigma_{\mathcal{R}}, \Sigma_E, \mathcal{R}, E, s, t, \ell_{\mathcal{R}}, \ell_E)$ where

- $\mathcal{R}$ is the finite set of N nodes representing resources and E is a set of arcs representing a handover of work relation;
- $\Sigma_{\mathcal{R}}$ and Σ_E are finite alphabets of the available vertex and arc labels, $s : E \to \mathcal{R}$ and $t : E \to \mathcal{R}$ are two maps indicating the source and target vertex of an arc;
- $\ell_{\mathcal{R}} : \mathcal{R} \to \Sigma_{\mathcal{R}}$ and $\ell_E : E \to \Sigma_E$ are two maps describing the labeling of the vertices and arcs.

In the following, for the sake of simplicity, we refer to a sociogram G as a tuple $G = (\mathcal{R}, E)$, with E as a multi-set of arcs. Each arc $e \in E$ will be shortly represented as a tuple $e = (r_i, r_j, (a_x, a_y, h_{xy}, q_{xy}))$ linking two nodes $r_i, r_j \in \mathcal{R}$, being r_i the source and r_j the target of e, and $(a_x, a_y, h_{xy}, q_{xy})$ the arc label. For convenience, we introduce $\xi_h(r_i, r_j, a_x) = \{h_{xy} : \exists a_y, \exists e = (r_i, r_j, (a_x, a_y, h_{xy}, q_{xy})) \in E\}$ which returns the values of handover of work between r_i (after a_x has been completed), and r_j, and $\xi_q(r_i, r_j, a_x) = \{q_{xy} : \exists a_y, \exists e = (r_i, r_j, (a_x, a_y, h_{xy}, q_{xy})) \in E\}$ which returns the numbers of times a_x has been executed by r_i. Furthermore, the set $G_{r_i}(a_x) = \{r' \in \mathcal{R} : \exists (r_i, r', (a_x, a_y, h_{xy}, q_{xy})) \in E\}$ is introduced to return the subset of resources for which an handover of work relation from r_i, after completing a_x, exists. The sociogram for the running example (Section 3) is shown in Fig. 2.

It is worth noting that the definition of sociogram we give is different from what is typically given in the organizational mining literature. Indeed, we refer to a labeled multidigraph with multiple edges between two nodes. This enables a greater expressiveness as we can encode not only the handover of generic work between two resources, but we also take into account which specific tasks one resource has handed over the other.

5. Resource replacement

This section is devoted to present the general methodology followed in this work. As shown in Fig. 3, we assume an organization relying on a Business Process Management system with monitoring capabilities of process execution. Data on executed activities are collected during the *process execution* phase and stored in an *event log*, which is then analyzed and elaborated through *organizational mining* techniques to derive the *sociogram* as described in Section 4. Furthermore, information is extracted from the system on the status of resource workload

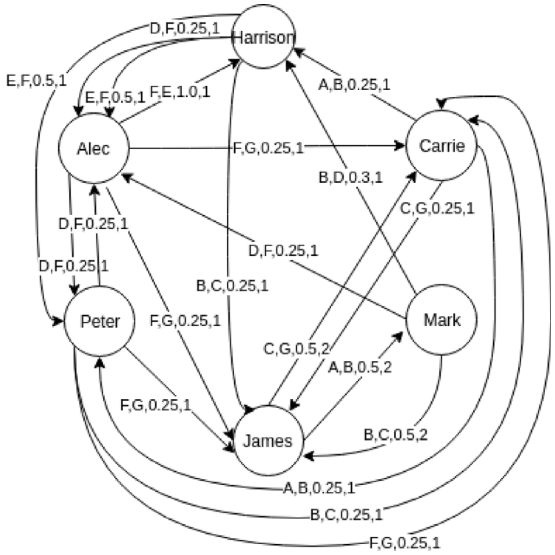


Fig. 2. Sociogram for the running example.

and priorities of activities (*run-time workload*). This information and the sociogram are used to drive the *resource replacement*. It is aimed to return the set of selected resources to replace the unavailable ones and their allocation to activities, taking into account the current load status of resources and domain-based constraints.

In the following, we describe the statement of RRP.

Statement of RRP Given a sociogram $G = (\mathcal{R}, E)$, let $R' \subseteq \mathcal{R}$ be a set of unavailable resources that must be replaced. For every unavailable resource $r_j \in R'$, let $T_{r_j} = \{a_1, \ldots, a_h\}$ be a multi-set of activities assigned to r_j that must be reassigned to some other resources. For every available resource $r_i \in \hat{R} = \mathcal{R} \setminus R'$, let γ_i be its workload factor, μ_i be its maximum workload factor and $\mathcal{A}(r_i)$ be its skills. The problem is to determine a set of resources $\{r_1, \ldots, r_n\} \subseteq \hat{R}$ that is collectively capable to replace resources in R' to perform their activities under a set of requirements, as follows:

- *affinity*: resources that are more compatible to those to replace are preferred. Affinity takes into account handover of work relations, capabilities of resources, performance and experience;
- *availability*: a resource can be selected only if its residual workload (i.e., the difference between the maximum and the current workloads) is enough to perform the requested activities;
- *load-balancing*: resources with a high residual workload are preferred over those with a low residual one, to improve load-balancing of resources;
- *priority*: in order to assign an activity with a given priority, all activities with a higher priority must be assigned.

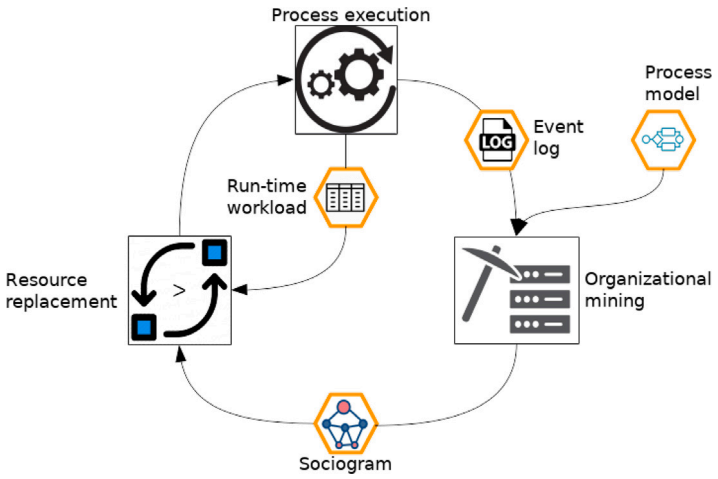


Fig. 3. Overview of the resource replacement methodology.

5.1. Cost factors

Given a resource $r_i \in \hat{R}$ which is a candidate to replace $r_j \in R'$ to perform the activity $a_k \in T_{r_j}$, we define a cost factor c_{ijk}, aimed at weighting the assignment of the activity to r_i based on its affinity with r_j and its residual workload $\mu_i - \gamma_i$. The cost factor c_{ijk} is defined as follows:

$$c_{ijk} = \psi(1 - sim(r_i, r_j, a_k)) + (1 - \psi)(1 - \frac{\mu_i - \gamma_i}{\mu_i}) \quad (1)$$

where $\psi \in [0, 1]$ weighs the contribution of affinity and workload in determining cost. In particular, the second component $(1 - \frac{\mu_i - \gamma_i}{\mu_i})$ is equal to 0 when the resource r_i is fully available (i.e., the current workload $\gamma_i = 0$). Conversely, when the resource is completely unavailable (i.e., $\gamma_i = \mu_i$), the second component of (1) is equal to 1. As for the first component of (1), $sim(r_i, r_j, a_k)$ measures the similarity between r_i and r_j as follows:

$$sim(r_i, r_j, a_k) = \omega_1 \cdot coll(r_i, r_j, a_k) + \omega_2 \cdot perf(r_i, r_j, a_k) + \omega_3 \cdot exp(r_i, r_j, a_k)$$

The function $sim(r_i, r_j, a_k)$ returns the degree of affinity between the two resources on the basis of (1) the collaborations that have been established to perform the activity a_k (i.e., $coll(r_i, r_j, a_k)$), (2) the speed with which a_k has been performed (i.e., $perf(r_i, r_j, a_k)$) and (3) the experience gained by the two resources in carrying out the activity (i.e., $exp(r_i, r_j, a_k)$). User parameters $\omega_1, \omega_2, \omega_3 \in [0, 1]$, such that $\omega_1 + \omega_2 + \omega_3 = 1$, are introduced to weigh the three similarity factors.

In details, $G_{r_i}(a_k)$ and $G_{r_j}(a_k)$ (see Section 4) are the sets of resources with whom respectively r_i and r_j have collaborated (through handover of work) to perform the activity a_k.

Let $\hat{h}(r_x, r_y, a_j) = \frac{1}{|\xi_h(r_x, r_y, a_j)|} \cdot \sum_{h_i \in \xi_h(r_x, r_y, a_j)} h_i$ be the average handover of work between r_x, executing a_j, and r_y. Finally, we introduce the function $\delta(x)$ which returns x if $x \leq 1$, 1 otherwise. The $coll$ function is defined as:

$$coll(r_i, r_j, a_k) = \begin{cases} \frac{1}{|G_{ij}(a_k)|} \cdot \sum_{r' \in G_{ij}(a_k)} \delta \frac{\hat{h}(r_i, r', a_k)}{\hat{h}(r_j, r', a_k)}, & \text{if } |G_{r_j}(a_k)| > 0 \\ \hat{h}(r_j, r_i, a_k), & \text{if } |G_{ij}(a_k)| = 0 \\ 0, & \text{otherwise.} \end{cases}$$

where $G_{ij}(a_k) = G_{r_i}(a_k) \cap G_{r_j}(a_k)$. The function $coll$ ranges in $[0, 1]$, returning 0 when the two resources do not share any collaborations, 1 if r_i interacts with all collaborators of r_j with the same handover of work. The $coll$ function is introduced under the assumption that replacing a resource with someone acquainted to work in the same team and with similar handover relations is preferable.

The $perf(r_i, r_j, a_k)$ function compares the time taken by the two resources to carry out a_k. Let t_{r_i} and t_{r_j} be the average time respectively taken by r_i and r_j to perform a_k, the $perf$ function is defined as

$$perf(r_i, r_j, a_k) = \delta\left(\frac{t_{r_j}}{t_{r_i}}\right).$$

The $perf$ function ranges in $[0, 1]$; values close to 0 mean that r_j is much faster than r_i. When r_i is on average faster than r_j, assigning a_k to r_i is also better than having it performed by r_j, hence the value of $perf$ is set to the maximum. The average times to perform an activity, if not available in the event log, can be estimated from the model by considering the difference in time between two causally related events.

Finally, the $exp(r_i, r_j, a_k)$ function compares the experience of the two resources to perform a_k, computed as the number of times the resource has carried out the activity ($q_{r_i}(a_k)$ and $q_{r_j}(a_k)$ respectively). The exp function is defined as

$$exp(r_i, r_j, a_k) = \delta\left(\frac{q_{r_i}}{q_{r_j}}\right),$$

where $q_{r'}(a_k) = \sum_{r_x \in G_{r'}(a_k)} \xi_q(r', r_x, a_k)$. This function ranges in $[0, 1]$. Values close to 0 mean that the experience of r_i is much smaller than that of r_j, while the function takes the maximum value when the experience of r_i is greater than or equal to that of r_j.

5.2. Priority of activities

As mentioned in the previous section, a requirement of the approach is to take into account priority of activities to replace. In this context, we model priority as a process-dependent function that defines a total order over the set of activities to replace. In particular, we assume that every activity has a priority value depending on (a) the process instance in which it is expected to be executed and (b) its position within the process. In particular:

- every process instance has a different priority, according to business rules. For such a reason, given two process instances X and Y with X having a higher priority, for all activities a_x in X and a_y in Y, priority of a_x is higher than that of a_y.
- every activity has a priority depending on its position within a process instance. Given a Casual Relation CR defined on a process schema, given two activities a_x, a_y belonging to the same process instance, such that $(a_x, a_y) \in CR$, then priority of a_x is higher than priority of a_y.

In the following we define priority as a function associating each activity to replace to a priority value.

Definition 8 (*Priority Function*)**.** Given a set of resources to replace $R' \subseteq \mathcal{R}$, and, for each $r_j \in R'$, the set of activities T_{r_j}, $P : \mathcal{R} \times \mathcal{A} \rightarrow \mathbb{N}$ is a bijective function mapping a pair $\langle r_j, a_k \rangle$ to a unique non-negative integer representing its priority.

As such, P produces a total order over the set of all pairs $\{\langle r_j, a_k \rangle : r_j \in R', a_k \in T_{r_j}\}$. The specific priority value, given to each process instance, is domain-dependent and therefore, left to implementation. Given a resource to replace $r_j \in R'$ and an activity to replace $a_k \in T_{r_j}$, in the following, we refer to P_k as to the set of pairs $p \in P$ with a priority higher than that of $\langle r_j, a_k \rangle$, i.e. $P_k = \{\langle r_x, a_y \rangle, r_x \in R', a_y \in T_{r_x} : P(\langle r_x, a_y \rangle) > P(\langle r_j, a_k \rangle)\}$.

6. An integer linear programming model

We mathematically formulate RRP through ILP with the aim of reassigning the activities of the unavailable resources at the minimum total cost. A penalty is also paid if an activity remains unassigned. The problem is modeled on the sociogram $G = (\mathcal{R}, E)$ by introducing the decision variable $x_{ijk}, \forall r_i \in \hat{R}, r_j \in R', a_k \in T_{r_j}$, that is equal to 1 if the resource r_i is selected for replacing the resource r_j in order to perform the activity a_k, 0 otherwise. For the sake of simplicity, in the following formulation, we refer each activity as well as each resource only through its subscript.

$$min\Big(\sum_{j \in R'} \sum_{k \in T_j} \sum_{i \in \hat{R}: a_{ik}=1} (c_{ijk} - M_k) x_{ijk}\Big) \tag{2}$$

$$\sum_{i \in \hat{R}: \alpha_{ik}=1} x_{ijk} \leq 1, \ \forall j \in R', k \in T_j \tag{3}$$

$$\sum_{j \in R'} \sum_{k \in T_j: \alpha_{ik}=1} \lambda_k x_{ijk} + \gamma_i \leq \mu_i, \forall i \in \hat{R} \tag{4}$$

$$\sum_{i \in \hat{R}: \alpha_{ik}=1} x_{ijk} \leq \frac{\sum_{i \in \hat{R}} \sum_{j' \in R'} \sum_{\langle r_{j'}, a_{k'} \rangle \in P_k: \alpha_{ik}=1} x_{ij'k'}}{|P_k|}, \forall j \in R', k \in T_j \tag{5}$$

$$x_{ijk} \in \{0, 1\} \tag{6}$$

The objective function (2) to minimize takes into account the total cost of assigning the activities of unavailable resources and the penalty (M_k) to pay for each activity a_k remained unassigned. Indeed, since we want to incentive the reassignments of all activities, in the objective function, the cost of each activity reassigned is reduced by the related penalty to pay in case it remains unassigned. It is worth noting that the specific value given to penalties is defined at application level and depends on the relative importance given to each activity to assign.

Constraints (3) impose that, for a given resource $r_j \in R'$, each activity is assigned to only one resource of $\hat{R}$. The parameter α_{ik} indicates that the resource $r_i \in \hat{R}$ is able to perform the activity $a_k \in T_{r_j}$. In particular, it takes the value 1 if $a_k \in \mathcal{A}(r_i)$, 0 otherwise. Constraints (4) assure that the availability requirement is satisfied. Indeed, the assignment of a set of activities to $r_i \in \hat{R}$ is possible only if the sum of the load requested for each of these activities (i.e., λ_k) and the current workload γ_i is less than or equal to the maximum workload μ_i. Constraints (5) guarantee that a lower priority activity cannot be assigned if all the higher priority activities are not assigned yet, thus satisfying the priority requirement. In particular, P_k denotes the set of activities with a priority greater than a_k. Indeed, the larger the instance to be solved, the greater the number of these constraints depending on both the number of resources to replace and the number of activities to assign. Finally, constraints (6) define the binary nature of the decision variables.

7. An LNS-based matheuristic

The solution of the ILP model, discussed in the previous section, may require a relatively high computational effort, especially when dealing with both medium-sized and large-sized instances of RRP. Therefore, in this section, we describe a Large Neighborhood Search (LNS) based matheuristic, designed to efficiently address both medium-sized and large-sized instances of the problem in reasonable computational times.

LNS is a metaheuristic proposed by Shaw (1998) and successfully used in several application contexts (Pisinger and Ropke, 2019). The main idea is that, starting from an initial solution, its quality is progressively improved through *destroy* and *repair* procedures (hereafter, referred as *moves*). The neighborhood to explore is then defined through the moves implemented in both the destroy and the repair phase. Then, the current solution is iteratively partially destroyed through the destroy moves and then, rebuilt by the repair ones. Usually, both the destroy and the repair moves are heuristic methods. For example, Åstrand et al. (2020) develop an LNS in order to quickly find high-quality schedules for the underground mine scheduling of mobile machines. Wolfinger (2021) show the good performances of the LNS designed for solving the Pickup and Delivery Problem with Time Windows, Split Loads and Transshipments, compared to other solution approaches. Finally, very recently, Şafak and Erdoğan (2023) propose an LNS for solving the Container Loading Problem and show the quality of its solutions, on a set of benchmark instances, compared to the state-of-the-art solution approaches.

However, LNS has been also used in a matheuristic framework, e.g., in Grenouilleau et al. (2020) for solving a home care scheduling with predefined visits and in Assunção and Mateus (2021), for addressing the Steiner team orienteering problem. In some works, the repair phase is fed to a solver for solving a mathematical programming formulation (e.g., Rastani and Çatay, 2023). In this work, we follow this option as detailed in Algorithm 1.

Algorithm 1 LNS-based matheuristic

Input:
initial temperature T_0,
number of removed assignments α,
cooling factor β,
total time limit TT,
time limit of each repair phase TR,
set of resources to replace R',
set of resource available $\hat{R}$,
set $\bar{T}$ containing, for each $j \in R'$, the set of its activities T_j
Output: best solution found sol^{best}

1: Current solution sol^{curr} :=***InitSol***$(R', \hat{R}, \bar{T})$;
2: Current temperature value $T^{curr} := T_0$;
3: Best solution $sol^{best} := sol^{curr}$;
4: **while** !***Stop***(TT) **do**
5: Solution sol':=***Destroy***(sol^{curr}, α);
6: sol' :=***Repair***(sol', TR);
7: **if** ***Cost***$(sol') <$ ***Cost***(sol^{curr}) **then**
8: $sol^{curr} := sol'$;
9: **if** ***Cost***$(sol') <$ ***Cost***(sol^{best}) **then**
10: $sol^{best} := sol'$;
11: **end if**
12: **else**
13: **if** ***Accept***(sol', T^{curr}) **then**
14: $sol^{curr} := sol'$;
15: $T^{curr} := \beta \times T^{curr}$;
16: **end if**
17: **end if**
18: **end while**

The proposed solution approach receives the set of resources to replace (R'), the set of available resource ($\hat{R}$) and the set of activities to be performed by each resource to be replaced ($\bar{T}$). Moreover, the algorithm takes as input an initial value of the temperature parameter

T_0 and the decreasing percentage of the temperature (i.e., the cooling factor) β, used in a Simulated Annealing based acceptance criterion. The parameters α and TR indicate the percentage of assignments to remove in the destroy phase and the maximum time given to the solver in the repair phase, respectively. The algorithm returns the best solution sol^{best} found within a maximum time limit TT chosen as a stopping criterion (Step 4). The initial solution is found through a greedy algorithm (Step 1) that tries to assign the activities of the unavailable resources to those resources that can perform them at the minimum cost. The initial solution found is feasible with regard to the constraints of the problem, i.e., according to load and priority conditions. In particular, Algorithm 2 details the routine ***InitSol()***. First, the set of possible triplets $\{\langle i,j,k\rangle |\ \forall i \in \hat{R}, j \in R', k \in T_j, \alpha_{ik} = 1\}$, is sorted by non-decreasing assignment costs (Step 1). At each iteration, the procedure adds a triplet to the initial solution Sol^0 only if it meets the load (4) and priority (5) constraints. These two conditions are verified by functions ***FeasibleLoad()*** and ***FeasiblePriority()***, respectively (Steps 10–14). If a triplet does not meet the criterion on load, it will be discarded. Instead, if it does not meet the criterion on priority, it will be put in the set T^{del} for being further verified (Step 16). Indeed, triples are chosen according to a cost criterion and, then, this order may not necessarily respect the priorities among activities. For instance, let us assume that (1) activity k_1 has a higher priority than k_2, (2) a triplet with k_2 has a lower cost than a triplet with k_1 and (3) the two triplets meet the load criterion. At the first execution of the loop (from Step 9 to Step 21), the triplet with k_2 will be put in T^{del} because it does not meet the priority constraint and the one with k_1 in Sol^0. Then, in the second execution of the loop, the triplet with k_2 will respect the priority constraint and will also be placed in Sol^0. The procedure ends when either no more feasible triplets can be added to the initial solution or there are no other triplets to be analyzed (Steps 22–23). It is worth remarking that $\boldsymbol{First}(T^{sort})$ (Steps 6 and 12) returns the triplet having the lowest assignment cost in the set T^{sort}.

Moreover, in order to improve the quality of the initial solution, we implement a local search that assures exploring a large neighborhood of it by randomly removing $2\times\alpha$ assignments. Then, a re-optimization is performed, with a time limit equal to $2\times TR$, by solving the mathematical model (2)–(6) in which only the decision variables associated with the maintained assignments are fixed. In each iteration of Algorithm 1, the procedure ***Destroy***(sol^{curr}, α) destroys the current solution sol^{curr} (Step 5). In other words, such a procedure selects a subset $\hat{A}$ of the triplets $\langle i,j,k\rangle$ such that the corresponding x-variables are equal to 1 in sol^{curr}. And, it sets to zero the x-variables of the triplets $\langle i,j,k\rangle \in \hat{A}$. The subset $\hat{A}$ is determined by invoking one of the following two remove moves: *cost_remove* and *random_remove*. The former removes α assignments chosen with a probability that depends on the assignment cost. In particular, the higher the assignment cost the higher the probability of being removed. Instead, the *random_remove* randomly chooses $2\times\alpha$ assignments to be removed. This move has been designed in order to possibly follow a new search direction and to properly shake the current solution. In each iteration, the remove move is randomly chosen. Then, the algorithm repairs the destroyed solution (Step 6) by solving the mathematical model (2)–(6) in which some decision variables are already fixed, corresponding to those that have not been removed in the destroy phase. In order to limit the computational time of the repair phase, the time limit TR, given to the solver, depends on the type of move chosen in Step 5 and in any case, it is only a very small percentage of the total time limit of the whole procedure. In particular, since in the *random_remove* move twice as many assignments are removed, the time limit given to the solver is double, i.e., $2\times TR$. In Section 8.3 we will show experimentally the validity of this choice. If the new solution found after a destroy and a repair move is better than the current one, the current solution is replaced by it (Steps 7–8). In addition, if the new solution is also better than the best one, the latter is replaced by it (Steps 9–10). Otherwise, the Accept procedure (Step 13), borrowed from the Simulated Annealing meta-heuristic, is applied in order to overcome local optimum (see Ropke and Pisinger, 2006 and Schrimpf et al., 2000). In particular, let val, f' and f be, respectively, a number randomly generated between 0 and 1, the objective function of the solution sol' and the best incumbent found so far. If $e^{-\frac{(f'-f)}{T^{curr}}} > val$, the new solution sol' is accepted and then, the best solution found so far sol^{curr} is replaced by it (Step 14). The value of the temperature $T^{curr} > 0$ is decreased every iteration through the cooling rate $\beta : 0 < \beta < 1$ (Step 15).

Algorithm 2 InitSol($R', \hat{R}, \bar{T}$)

Input:
set of resources to replace R',
set of resources available $\hat{R}$,
set $\bar{T}$ containing, for each $j \in R'$, the set of its activities T_j
Output: initial solution sol^0

1: Let $T^{sort} := \{\langle i,j,k\rangle |\ \forall i \in \hat{R}, j \in R', k \in T_j, \alpha_{ik} = 1\}$ be the set of triplets sorted by non-decreasing assignment costs c_{ijk}
2: Let $\tilde{\mu_i} := \mu_i - \gamma_i$ be the residual workload $\forall i \in \hat{R}$
3: $Sol^0 := \emptyset$
4: $Exit := FALSE$
5: **while** $!Exit$ **do**
6: $\langle i*, j*, k*\rangle := \boldsymbol{First}(T^{sort})$
7: $Found := FALSE$
8: $T^{del} := \emptyset$
9: **while** $\langle i*, j*, k*\rangle \neq null$ **do**
10: **if** ***FeasibleLoad***$(\langle i*, j*, k*\rangle)$ **then**
11: **if** ***FeasiblePriority***$(\langle i*, j*, k*\rangle)$ **then**
12: $\tilde{\mu_{i*}} := \tilde{\mu_{i*}} - \lambda_{k*}$
13: $Sol^0 := Sol^0 \bigcup \{\langle i*, j*, k*\rangle\}$
14: $Found := TRUE$
15: **else**
16: $T^{del} := T^{del} \bigcup \{\langle i*, j*, k*\rangle\}$
17: **end if**
18: **end if**
19: $T^{sort} := T^{sort} \setminus \langle i*, j*, k*\rangle$
20: $\langle i*, j*, k*\rangle := \boldsymbol{First}(T^{sort})$
21: **end while**
22: **if** $(!Found \vee T^{del} = \emptyset)$ **then**
23: $Exit := TRUE$
24: **end if**
25: $T^{sort} := T^{del}$
26: **end while**

8. Experiments

This section describes the experimentations that have been carried out to evaluate the performance of the proposed approach:

- the first experiment, discussed in Section 8.1, is focused on the evaluation of the execution time for the sociogram creation;
- the second set of experiments, discussed in Section 8.2, are aimed to provide a comparison between the ILP model and the LNS-based matheuristic for medium-sized and large-sized problems.

8.1. Evaluation of sociogram creation

In this section, we refer to the real-life dataset published for the BPI Challenge 2012 (https://www.win.tue.nl/bpi/doku.php?id=2012:challenge), that was generated by a Dutch Financial Institute. The event log contains 262 200 events in 13 087 traces referring to an application process for personal loan, recorded from October 1st, 2011 to March 14th, 2012.

As mentioned before, the methodology proposed in this paper requires a conform event log (i.e., such that all traces are 100% fitting with the model). Given that the model was not available with the dataset, a process model has been discovered through the Heuristic Miner (Weijters et al., 2006) plugin in ProM (Van Dongen et al., 2005).

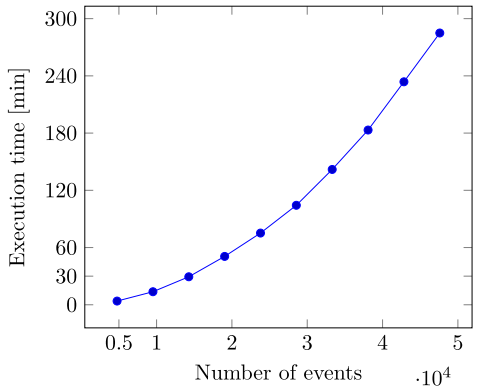


Fig. 4. Sociogram creation time for event logs with increasing number of events.

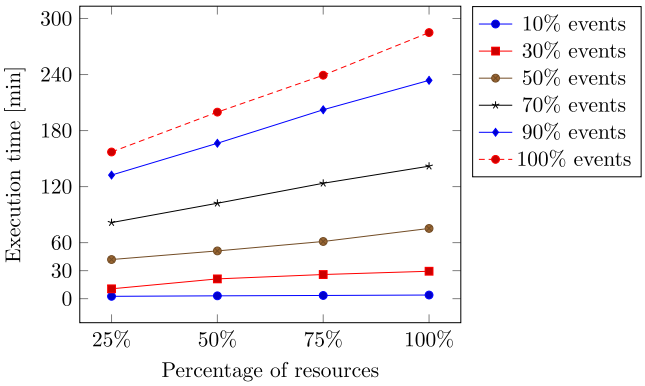


Fig. 5. Sociogram creation time (in minutes) for event logs with an increasing number of resources (from 25% to 100% of the original event log), for logs with various sizes.

Then, a preliminary conformance checking step has been performed and non-fitting traces have been filtered out, resulting in a number of 7974 traces and 47 634 events after the filtering. Each trace includes an average number of 6 events, ranging from a minimum of 3 to a maximum of 41 and corresponding to 8 activity names. Finally, the total number of resources are 62.

In the following, we report an experiment aimed at analyzing efficiency in terms of running time needed for the creation of the sociogram. The experiment has been carried out on Linux Ubuntu 18.04 LTS 64-bit running on a 4-core 2.2 GHz processor with 16 GB RAM. The sociogram creation has been implemented in Java.

8.1.1. Experimental setup

The experiment has been performed by analyzing how the sociogram creation time varies with (1a) the size of the event log and (1b) the number of resources in a log.

As for test (1a), starting from the dataset D obtained after the filtering, 9 additional datasets ($D_1, \ldots, D_9$) have been generated, each containing a given percentage of the overall events, ranging from 10% to 90%. The selection has been performed randomly for each dataset, avoiding breaking traces. On the other hand, for the test (1b), we have generated for each D_i three additional datasets $D_{i,25}$, $D_{i,50}$ and $D_{i,75}$, each containing respectively the 25%, 50% and 75% of resources in D_i. These datasets have been generated by artificially reducing the number of resources but keeping the same number of traces and events.

8.1.2. Results and discussion

Results of test (1a) are shown in Fig. 4, where the time to build the sociogram for datasets $D_1, \ldots, D_9$ and D are reported. A quadratic trend (with correlation R=0.99) can be clearly identified. This is due to the time needed to align each trace to the model, after which the sociogram can be computed in $\mathcal{O}(n)$ steps, where n is the number of events in the log.

In Fig. 5, we represent the sociogram creation time for several datasets, each containing an increasing number of resources (i.e., percentage of resources with regard to the original event log). For each test, an almost linear (or even slightly sublinear in some cases) trend can be identified.

According to the above results, it is evident that the creation of the sociogram is a demanding phase. It is worth noting that, while the resource replacement phase is to be executed at need, a sociogram should only be updated when the model of interaction of resources changes. This means that, since the event log grows incrementally (traces can only be added), it is possible to devise a refresh procedure for a sociogram that takes into account only the most recent traces, i.e. those generated after the last refresh. As such, the sociogram model can be updated incrementally on a regular basis or at need.

8.2. Comparison between the ILP model and LSN-based matheuristic

8.2.1. Experimental setup

The experiment is based on the execution of the ILP model and the LSN-based matheuristic on a set of problem instances of increasing size, synthetically generated from the same dataset.

The dataset is inspired to the case study and consists of a log including 1040 traces and 7280 events. Each trace includes 7 events on average, ranging from 6 to 8 and representing 7 distinct activities. The number of resources is 120. The reason for introducing an additional event log is that the dataset used in Section 8.1 cannot be directly used to evaluate the resource replacement approach, as it does not contain relevant information (e.g., workload, missing resources). Moreover, we want to analyze a realistic scenario of a medium–large company, where the number of resources and number of activities to be performed by each resource are higher than the values reported in such an event log. Furthermore, our goal is to compare the ILP model and the LNS-based matheuristic as the complexity varies. We would like to point out that although the chosen dataset is synthetic, this does not affect the significance of the results and their applicability to a real context. In fact, the problem instance depends on the sociogram rather than on the chosen process model. For each causal relation (a_i, a_j) in the process model, there will be as many arcs in the sociogram as for each pair of resources assigned to a_i and a_j respectively. Thus, even simple processes and logs with few traces can generate complex problem instances. The size of the log only affects the accuracy with which the sociogram weights are calculated.

Given the set of resources $\mathcal{R}$, the set $\mathcal{A}$ of activities, a number N_r of resources to replace, and a number N_a of activities for each missing resource, a problem instance is generated by a problem instance creator script through the following procedure:

- initialize the list of activities and list of resources from the log;
- randomly select N_r resources to replace from the resource list. The remaining ($|\mathcal{R}| - N_r$) resources are considered available;
- for each selected resource:
  - randomly select N_a activities from the activity list. The selection can contain duplicated activities, that represent multiple occurrences of the same activity in the same or in different process instances;
  - for each selected activity to replace: assign it a random priority from $[0, N_r * N_a)$ that has not been previously assigned (each activity to replace will have a unique priority number able to define a total order over the set of activities to replace).

The problem instance was then created by defining a variable for each available resource r and each activity a to replace that is

Table 3
Problem instances features.

Problem instance	N_r	N_a	Total possible assignments
S1, S2	10	10	100
S3, S4	10	15	150
S5, S6	15	10	150
S7, S8	15	15	225
M1, M2	20	20	400
M3, M4	20	25	500
M5, M6	25	20	500
M7, M8	20	30	600
M9, M10	30	20	600
L1, L2	25	25	625
L3, L4	30	25	750
L5, L6	25	30	750
L7, L8	30	30	900

compatible with r, i.e., $a \in \mathcal{A}(r)$. Factors such as capability of resources and average load request for activities were extracted from the event log. Cost factors c were pre-computed from the sociogram and the log for any triple (i-th available resource, jth resource to replace, kth activity to replace) to speed up the problem instance generation process. Parameter ω_1 was set to 0.5, while parameters ω_2 and ω_3 were set to 0.25 to weight more the collaboration factor. Parameter ψ was set to 0.5. It should be noted that the sociogram is calculated only once and is not dependent on problem instances. The maximum workload was set to 1 for all resources, whereas the current workload factor for each resource was derived from the event log.

Following the above procedure, we generated a number of problem instances by varying parameters N_r and N_a from 20 to 30 with step 5. Therefore, the number of total possible assignments, given by $N_r * N_a$, ranges from 400 to 900. We generated two problem instances for each pair N_r, N_a, resulting in a total of 18 problem instances as summarized in Table 3. Problem instances were classified as medium-sized if the number of possible assignments is up to 600 ($M_1 - M_{10}$), large-sized otherwise ($L_1 - L_8$). The choice of this problem instance size is motivated by the fact that the ILP model can solve RRP to optimality on small-sized problem instances, up to 150–200 total possible assignments. However, on larger problem instances, it usually does not terminate within the CPU time limit, which was set to 2 h. For the sake of completeness, we also generated a set of small-sized instances with $N_r, N_a \in \{10, 15\}$ ($S_1 - S_8$).

For each problem instance, the model was solved with a time of 7200 s and the penalty M_k was set to 100 for each activity. As for the matheuristic, we used ParamILS (Hutter et al., 2009) to automatically configure the parameters. The range of values tested for each parameter is: α from 0.2 to 0.7 with step 0.05, $TR \in \{25, 50, 100, 150, 200\}$, T_0 from 50 to 500 with step 50 and β from 0.01 to 0.05 with step 0.01. We ran ParamILS on the three sizes of problem instances independently, obtaining as the best configuration: $\alpha = 0.45$, $T_0 = 300$ and $\beta = 0.03$ for each type of instance; $TR = 100$ for large-sized problem instances and $TR = 50$ for small-sized and medium-sized problem instances. We set $TT = 7200$ for all instances for comparison purposes.

The experiments were carried out on a 4-core 2.39 GHz processor with 32 GB RAM. The matheuristic was implemented in Java, and the ILP model was solved through IBM ILOG CPLEX API (version 20.1).

8.2.2. Results and discussion

In order to compare the performances of the two proposed approaches on different instances,the results are presented in terms of gain of the matheuristic over the model on the problem instance i as:

$$gain(i) = \frac{FO_{mh}(i) - FO_m(i)}{min(FO_{mh}(i), FO_m(i))} \cdot 100$$

where $FO_{mh}(i)$ and $FO_m(i)$ are the values of the objective function achieved on the problem instance i by the matheuristic and the ILP model, respectively. Positive values of gain correspond to an advantage of the matheuristic over the model, and the gain gives the percentage of objective function reduction achieved by it. Vice versa, negative values indicate a loss in performance of the matheuristic, and the gain gives the percentage of objective function reduction achieved by the model.

Table 4
Comparison between the ILP model and the matheuristic.

Problem instance	Gap	$gain_{best}$	$gain_{avg}$	$timeToBest_{avg}$ (s)
S1	0.000	0.000	0.000	2.24
S2	0.000	0.000	0.000	2.39
S3	0.000	0.000	0.000	6.11
S4	0.000	0.000	0.000	7.56
S5	0.000	0.000	0.000	3.67
S6	0.000	0.000	0.000	3.87
S7	0.000	0.000	0.000	10.88
S8	0.000	0.000	0.000	11.10
M1	0.050	0.000	0.000	731.92
M2	0.008	0.000	0.000	482.86
M3	0.001	0.000	0.000	2,803.79
M4	0.001	0.000	0.000	476.66
M5	0.032	0.003	0.001	4,379.76
M6	0.050	0.000	0.000	763.47
M7	0.054	0.002	0.000	5,784.27
M8	0.036	0.002	0.002	2,722.41
M9	0.189	0.001	0.001	4,215.34
M10	0.093	1.238	1.238	3,932.59
L1	0.054	1.143	1.143	2,454.41
L2	0.032	0.001	−0.046	5,579.87
L3	0.294	1.126	1.125	5,348.27
L4	0.248	0.278	0.278	5,707.17
L5	0.104	0.591	0.589	5,792.46
L6	0.133	0.004	0.004	5,230.81
L7	0.297	3.845	3.738	6,226.43
L8	0.221	1.032	0.561	5,312.98

Table 4 reports gains computed by considering both the average values of the objective function achieved by the matheuristic ($gain_{avg}$) and the best value ($gain_{best}$) over 10 runs. The table also reports the *Gap* produced, in the time limit of 7200 seconds, by CPLEX for the ILP model. Finally, the time in seconds required by the matheuristic to reach the best solution averaged over the 10 runs ($timeToBest_{avg}$) is also shown.

From Table 4, it turns out that the advantage of the matheuristic grows as the complexity of the problem instance increases. First, one can note that our matheuristic always finds the optimal solution, certificated by CPLEX solving the ILP model, on the small-sized instances. On this set of instances, our solution approach requires an average time to best that is comparable to the time needed to CPLEX (5.98 s of the former versus 5.57 s of the latter). Second, it is worth noting that the ILP model is never solved to optimality by CPLEX even in 7200 seconds on all the other problem instances, i.e., the medium-sized and large-sized ones. In particular, on the medium-sized problem instances, the average Gap is equal to 0.051, whereas, on the large-sized problem instances, it is 0.173. Indeed, on average, the gain for large-sized problem instances (0.924 for $gain_{avg}$ and 1.002 for $gain_{best}$) is greater than the gain achieved on medium-sized problem instances (0.124 for $gain_{avg}$ and 0.125 for $gain_{best}$). The increase in gain with respect to the number of possible assignments is even more evident in Fig. 6, where the gains of problem instances with same N_a and N_b have been averaged. It can be seen that up to about 750 total possible assignments, the two curves are almost identical; then, they differ markedly. This indicates that the variability of solutions up to 750 is low and solutions are independent of the run, and, therefore, of the initialization. Clearly, for more complex situations, where the solution space is larger, the greedy heuristic fails to determine a good quality initial solution. Hereafter, we focus only on medium-sized and large-sized problem instances, where CPLEX fails to certify the optimum.

It is noteworthy that in the 6 cases where the two methods tie (5 if $gain_{best}$ is considered), the matheuristic obtains the best result in

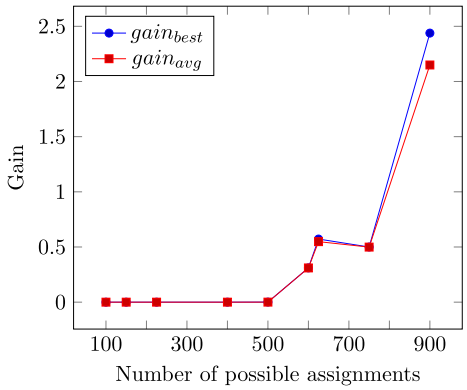


Fig. 6. Gain Vs. number of possible assignments.

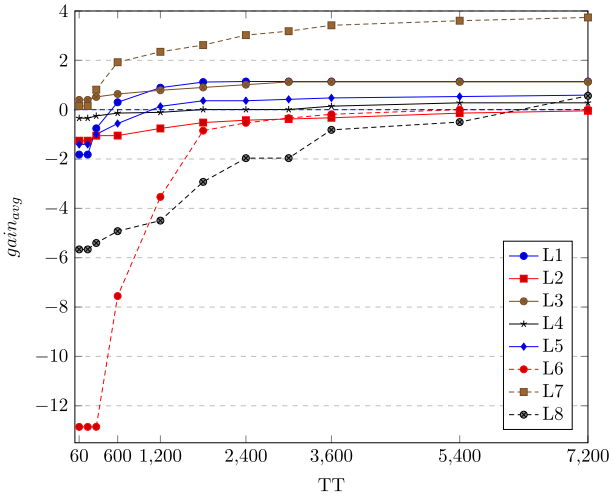


Fig. 7. $gain_{avg}$ of the matheuristic over large-sized problem instances, varying TT.

by far less time than the model. Indeed, in these cases, the average time to achieve the best result ranges from 6.67% to 38.89% of that required by the ILP model, except for problem instance M7 where it is 5784 seconds. Fig. 7 shows the $gain_{avg}$ achieved by the matheuristic over large-sized problem instances as it varies with the time limit (TT). Whatever TT, in these experiments, the $gain_{avg}$ is calculated by considering a time limit of 7200 s for the model, thus penalizing the matheuristic. In about 63% of cases, the matheuristic achieves a positive gain in a maximum time of 30 min.

In Table 5, we report the number of activities the model was able to assign to available resources, and the number of assignments returned by the matheuristic averaged over the 10 runs with also standard deviations reported in brackets. In most of the medium-sized problem instances, the two approaches return the same number of assignments. Instead, in most of the large-sized problem instances, the matheuristic is able to allocate more activities than the model. Clearly, on L_2, where the matheuristic performs worse on average, the number of assignments in the solution found by the model is greater than that in the solution of the matheuristic.

The Wilcoxon Signed Rank Sum Test (Wilcoxon, 1945) is performed to check the null hypothesis that neither approach is better than the other in terms of objective function on medium-sized and large-sized instances. The test is performed both on $gain_{avg}$ and $gain_{best}$. In the worst-case scenario, where we consider $gain_{avg}$, the two approaches return the same results on 6 problem instances and the matheuristic outperforms the model on 11 of 12 remaining problem instances. When $gain_{best}$ is taken into account, the matheuristic performs better than the

Table 5
Number of assignments in the solutions provided by the two approaches.

Problem instance	ILP Model No. of assignments	LNS-based Matheuristic Average no. of assignments
M1	326	326.0 (0.00E+00)
M2	301	301.0 (0.00E+00)
M3	478	478.0 (6.10E−06)
M4	361	361.0 (0.00E+00)
M5	420	420.0 (0.00E+00)
M6	426	426.0 (0.00E+00)
M7	459	459.0 (0.00E+00)
M8	463	463.0 (0.00E+00)
M9	381	381.0 (0.00E+00)
M10	398	403.0 (0.00E+00)
L1	436	441.0 (0.00E+00)
L2	420	419.8 (6.00E−01)
L3	352	356.0 (0.00E+00)
L4	361	365.0 (0.00E+00)
L5	347	349.0 (0.00E+00)
L6	314	316.0 (0.00E+00)
L7	352	365.6 (8.00E−01)
L8	386	388.2 (1.94E+00)

model on 13 problem instances and returns the same value of the model in the remaining 5 problem instances. The null hypothesis is rejected in both cases, with significance levels of 99.25% (p-value = 0.0076) and 99.85% (p-value = 0.0015), for the average and the best gains, respectively.

Some managerial insights can be also provided from the solutions obtained. Regarding the activity to reassign, the *percentage replacement degree* (i.e., the ratio between the number of reassigned activities and the number of activities to replace) can be computed, giving us a measure of the quality of the solutions found. For example, considering the instance named $M1$, the *percentage replacement degree* for each activity is on average equal to 81.50% meaning that almost all the activities have been reassigned to the available resources (i.e., 326 activities over 400). Moreover, on this instance, one can also observe that all the 326 reassigned activities have higher priority than those that have not been reassigned. Regarding the resources available to replacements, one can compute the *residual workload* for each of them after the new reassignments are made. This way, the manager can easily identify the most critical resources (i.e., those that are used for almost all of their residual workload) and then he/she can decide whether it is appropriate to invest more in them (e.g., to pay overtime hours) or in acquiring new resources with the same skills.

8.3. Sensitivity analysis on the moves of the LNS-based matheuristic

This section aims at discussing a sensitivity analysis on the designed moves in the LNS-based matheuristic. First, we analyze the solutions obtained with only one move at time, compared to the solutions detected when both the moves are used. To this purpose, we executed 10 run for each of the two cases on all the medium-sized and the large-sized instances.

The results confirm that using both the moves instead of only one of the two helps finding better solutions on average. In particular, on the medium-sized instances the average of the gains when only the *cost_remove* move and only the *random_remove* move are used in the destroy step are 0.67% and 0.42%, respectively. These gains become higher on the large-sized instances, being 1.99% and 3.04% respectively with only the *cost_remove* move and the *random_remove* move.

Another performed analysis is related to the two input parameters of the *random_remove* move, i.e., the number of assignments to remove and the time to repair after this move is applied. In the proposed approach, they are respectively 2α and $2TR$. It means that we are always removing twice as many assignments as removed in the *cost_remove* move, and

we are always giving the solver twice as much time to repair the destroyed solution. This choice is first motivated by the fact that the *random_remove* move was designed with the aim of shaking as much as possible the current solution. To experimentally evaluate our design choice, we analyze the sensitivity of the solutions by varying these values. For this reason, we consider also these three possible cases in the *random_remove* move:

- removing α assignments, with a time to repair equals to TR (case denoted as (α, TR));
- removing 2α assignments, with a time to repair equals to TR (case denoted as $(2\alpha, TR)$);
- removing α assignments, with a time to repair equals to 2TR (case denoted as $(\alpha, 2TR)$);

We compare each of them with the base case, i.e., the case in which we remove 2α assignments with a time to repair equals to 2TR, i.e., $(2\alpha, 2TR)$. For each instance and for each of these three cases, we compute the average gain with regard the base case. Then, we average the gains on the medium-sized instances and on the large-sized instances. The computational results confirm that the base case outperforms the other three cases. In particular, on the medium-sized instances, the average gains are not very significant, being $0.006\%, 0.099\%$ and 0.002%, respectively, with $(\alpha, TR), (2\alpha, TR), (\alpha, 2TR)$. Instead, these gains become more significant considering the large-sized instances on which they are $0.079\%, 0.052\%$ and 0.040%, respectively, with (α, TR), $(2\alpha, TR), (\alpha, 2TR)$. In the experiments with large-sized instances, we note that the worst configuration is (α, TR), indicating that an increase in both TR and alpha is useful in the *random_remove* move. Furthermore, whatever the instance size, the results indicate that when we remove twice as many assignments as the *cost_remove* move, the same TR as in *cost_remove* move is not sufficient. As a matter of fact, considering only the times taken by the *random_remove* move in the base case, it can be seen that on average it takes 199.48 s on the large-sized instances and 95.83 s on the medium-sized instances, confirming that a double TR is necessary. Indeed, after the *random_remove* move, on average, only 10.19% and 2.13% of the ILP models are solved to optimality, respectively, considering medium-sized and large-sized instances.

9. Conclusion

In this work, we proposed a data-driven approach to support resource replacement in organizations, based on a notion for handover of work that is capable to model collaboration among resources on the basis of causal relations between activities. The solution of such a problem becomes crucial in order to guarantee continuity in the business process. The proposed approach exploits event logs of past process executions and takes into account also run-time resource workload and load-balancing. Although we referred mostly to human-centric processes in both the case study and the experimentation, the approach is general enough to be applied in cases where a mix of users and machines cooperate together, provided additional constraints on which resources cannot be selected for replacement. Then, we formulated the Resource Replacement Problem (RRP) through ILP, in order to select the available resources for performing the activities already assigned to the unavailable resources, at the minimum total cost, under both the workload and the priority constraints. In order to efficiently address RRP on medium-sized and large-sized problem instances, we also designed a Large Neighborhood Search based matheuristic. Experimental results, obtained on a set of real-alike problem instances, showed the effectiveness of the proposed matheuristic. In fact, it outperforms the ILP model on average, even concerning the solution quality in the cases in which the latter reaches the CPU time limit given to the solver (i.e., two hours). In addition, the sensitivity analysis performed on the moves designed in the proposed matheuristic confirmed the need of having all of them for detecting good quality solutions.

As future work, we plan to extend the sociogram to take into account further social relations among resources, that can be retrieved from event logs, e.g., subcontracting or cooperation, as well as from other data sources, e.g., enterprise social networks. We also plan to develop further applications, including analyses of what-if scenarios and evaluation of robustness to critical situations, to provide a measure of organization resilience.

CRediT authorship contribution statement

Claudia Diamantini: Conceptualization, Investigation, Methodology, Software, Validation, Writing – review & editing. **Ornella Pisacane:** Conceptualization, Investigation, Methodology, Software, Validation, Writing – review & editing. **Domenico Potena:** Conceptualization, Investigation, Methodology, Software, Validation, Writing – review & editing. **Emanuele Storti:** Conceptualization, Investigation, Methodology, Software, Validation, Writing – review & editing.

Data availability

Data will be made available on request.

Acknowledgments

The authors wish to thank Marco Ciotti and Michele Florio for their support in the implementation and experimentation of the approach. The work was partially founded by Università Politecnica delle Marche, under RSA-B 2020 project.

References

Appice, A., 2018. Towards mining the organizational structure of a dynamic event scenario. J. Intell. Inf. Syst. 50 (1), 165–193. http://dx.doi.org/10.1007/s10844-017-0451-x.

Arias, M., Munoz-Gama, J., Sepúlveda, M., Miranda, J.C., 2018. Human resource allocation or recommendation based on multi-factor criteria in on-demand and batch scenarios. Eur. J. Ind. Eng. 12 (3), 364–404. http://dx.doi.org/10.1504/EJIE.2018.092009.

Assunção, L., Mateus, G.R., 2021. Coupling feasibility pump and large neighborhood search to solve the steiner team orienteering problem. Comput. Oper. Res. 128, 105175.

Åstrand, M., Johansson, M., Zanarini, A., 2020. Underground mine scheduling of mobile machines using constraint programming and large neighborhood search. Comput. Oper. Res. 123, 105036.

Burattin, A., Sperduti, A., Veluscek, M., 2013. Business models enhancement through discovery of roles. In: 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), Singapore, April 16-19. IEEE, pp. 103–110. http://dx.doi.org/10.1109/CIDM.2013.6597224.

Cabanillas, C., García, J.M., Resinas, M., Ruiz, D., Mendling, J., Cortés, A.R., 2013. Priority-based human resource allocation in business processes. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (Eds.), Service-Oriented Computing - 11th International Conference, ICSOC 2013, Berlin, Germany, December 2-5. In: Lecture Notes in Computer Science, Vol. 8274, Springer, pp. 374–388. http://dx.doi.org/10.1007/978-3-642-45005-1_26.

Conforti, R., de Leoni, M., La Rosa, M., van der Aalst, W.M., ter Hofstede, A.H., 2015. A recommendation system for predicting risks across multiple business process instances. Decis. Support Syst. 69, 1–19. http://dx.doi.org/10.1016/j.dss.2014.10.006.

De Bruecker, P., Van den Bergh, J., Beliën, J., Demeulemeester, E., 2015. Workforce planning incorporating skills: State of the art. European J. Oper. Res. 243 (1), 1–16. http://dx.doi.org/10.1016/j.ejor.2014.10.038.

Diamantini, C., Genga, L., Potena, D., van der Aalst, W., 2016. Building instance graphs for highly variable processes. Expert Syst. Appl. 59, 101–118. http://dx.doi.org/10.1016/j.eswa.2016.04.021.

Grenouilleau, F., Lahrichi, N., Rousseau, L.-M., 2020. New decomposition methods for home care scheduling with predefined visits. Comput. Oper. Res. 115, 104855.

Havur, G., Cabanillas, C., Mendling, J., Polleres, A., 2016. Resource allocation with dependencies in business process management systems. In: La Rosa, M., Loos, P., Pastor, O. (Eds.), Business Process Management Forum: BPM Forum 2016, Rio de Janeiro, Brazil, September 18-22. Springer International Publishing, pp. 3–19. http://dx.doi.org/10.1007/978-3-319-45468-9_1.

Hirsch, M.J., Ortiz-Peña, H., 2017. Information supply chain optimization with bandwidth limitations. Int. Trans. Oper. Res. 24 (5), 993–1022. http://dx.doi.org/10.1111/itor.12392.

Huang, Z., Lu, X., Duan, H., 2012. A task operation model for resource allocation optimization in business process management. IEEE Trans. Syst. Man Cybern. A 42 (5), 1256–1270. http://dx.doi.org/10.1109/TSMCA.2012.2187889.

Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T., 2009. ParamILS: an automatic algorithm configuration framework. J. Artificial Intelligence Res. 36 (1), 267–306. http://dx.doi.org/10.5555/1734953.1734959.

Kumar, A., van der Aalst, W.M.P., Verbeek, H.M.W., 2002. Dynamic work distribution in workflow management systems: How to balance quality and performance. J. Manage. Inf. Syst. 18 (3), 157–194. http://dx.doi.org/10.1080/07421222.2002.11045693.

Lee, J., Lee, S., Kim, J., Choi, I., 2019. Dynamic human resource selection for business process exceptions. Knowl. Process Manag. 26 (1), 23–31. http://dx.doi.org/10.1002/kpm.1591.

Liu, Y., Wang, J., Yang, Y., Sun, J., 2008. A semi-automatic approach for workflow staff assignment. Comput. Ind. 59 (5), 463–476. http://dx.doi.org/10.1016/j.compind.2007.12.002.

Martin, N., Depaire, B., Caris, A., Schepers, D., 2020. Retrieving the resource availability calendars of a process from an event log. Inf. Syst. 88, 101463. http://dx.doi.org/10.1016/j.is.2019.101463.

Peterson, J.L., 1977. Petri nets. ACM Comput. Surv. 9 (3), 223–252. http://dx.doi.org/10.1145/356698.356702.

Pika, A., Leyer, M., Wynn, M.T., Fidge, C.J., Hofstede, A.H.T., van der Aalst, W.M.P., 2017. Mining resource profiles from event logs. ACM Trans. Manag. Inf. Syst. 8 (1), 1–30. http://dx.doi.org/10.1145/3041218.

Pisinger, D., Ropke, S., 2019. Large neighborhood search. In: Gendreau, M., Potvin, J.-Y. (Eds.), Handbook of Metaheuristics. Springer International Publishing, pp. 99–127. http://dx.doi.org/10.1007/978-3-319-91086-4_4".

Pufahl, L., Ihde, S., Stiehle, F., Weske, M., Weber, I., 2021. Automatic resource allocation in business processes: A systematic literature survey. Comput. Res. Repository arXiv:2107.07264.

Rastani, S., Çatay, B., 2023. A large neighborhood search-based matheuristic for the load-dependent electric vehicle routing problem with time windows. Ann. Oper. Res. 324 (1), 761–793. http://dx.doi.org/10.1007/s10479-021-04320-9.

Ropke, S., Pisinger, D., 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp. Sci. 40 (4), 455–472.

Şafak, O., Erdoğan, G., 2023. A large neighbourhood search algorithm for solving container loading problems. Comput. Oper. Res. 154, 106199.

Schönig, S., Cabanillas, C., Jablonski, S., Mendling, J., 2016. A framework for efficiently mining the organisational perspective of business processes. Decis. Support Syst. 89, 87–97.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., Dueck, G., 2000. Record breaking optimization results using the ruin and recreate principle. J. Comput. Phys. 159 (2), 139–171. http://dx.doi.org/10.1006/jcph.1999.6413.

Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M., Puget, J.-F. (Eds.), Principles and Practice of Constraint Programming—CP98: 4th International Conference, Pisa, Italy, October 26–30. Springer Berlin Heidelberg, pp. 417–431. http://dx.doi.org/10.1007/3-540-49481-2_30.

Song, M., van der Aalst, W.M., 2008. Towards comprehensive support for organizational mining. Decis. Support Syst. 46 (1), 300–317. http://dx.doi.org/10.1016/j.dss.2008.07.002.

Van Der Aalst, W.M., Reijers, H.A., Song, M., 2005. Discovering social networks from event logs. Comput. Support. Coop. Work (CSCW) 14 (6), 549–593. http://dx.doi.org/10.1007/s10606-005-9005-9.

van Dongen, B.F., van der Aalst, W.M.P., 2004. Multi-phase process mining: Building instance graphs. In: Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T.-W. (Eds.), Conceptual Modeling – ER 2004. Springer Berlin Heidelberg, pp. 362–376. http://dx.doi.org/10.1007/978-3-540-30464-7_29.

Van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H., Weijters, A., van Der Aalst, W.M., 2005. The ProM framework: A new era in process mining tool support. In: Ciardo, G., Darondeau, P. (Eds.), Applications and Theory of Petri Nets 2005: 26th International Conference, ICATPN 2005, Miami, USA, June 20-25. Springer, pp. 444–454. http://dx.doi.org/10.1007/11494744_25.

van Hulzen, G., Martin, N., Depaire, B., 2021. Looking beyond activity labels: Mining context-aware resource profiles using activity instance archetypes. In: Polyvyanyy, A., Wynn, M.T., Van Looy, A., Reichert, M. (Eds.), Business Process Management Forum. Springer International Publishing, pp. 230–245. http://dx.doi.org/10.1007/978-3-030-85440-9_14.

Wasserman, S., Faust, K., 1994. Social Network Analysis: Methods and Applications. In: Structural Analysis in the Social Sciences, Vol. 8, Cambridge University Press, http://dx.doi.org/10.1017/CBO9780511815478.

Weijters, A., van Der Aalst, W.M., De Medeiros, A.A., 2006. Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven, Tech. Rep. WP, Vol. 166, pp. 1–34.

Wilcoxon, F., 1945. Individual comparisons by ranking methods. Biom. Bull. 1 (6), 80–83. http://dx.doi.org/10.2307/3001968.

Wolfinger, D., 2021. A large neighborhood search for the pickup and delivery problem with time windows, split loads and transshipments. Comput. Oper. Res. 126, 105110.

Xie, Y., Chien, C.-F., Tang, R.-Z., 2016. A dynamic task assignment approach based on individual worklists for minimizing the cycle time of business processes. Comput. Ind. Eng. 99, 401–414. http://dx.doi.org/10.1016/j.cie.2015.11.023.

Yang, H., Wang, C., Liu, Y., Wang, J., 2008. An optimal approach for workflow staff assignment based on hidden Markov models. In: Meersman, R., et al. (Eds.), On the Move to Meaningful Internet Systems: OTM 2008 Workshops. Springer Berlin Heidelberg, pp. 24–26. http://dx.doi.org/10.1007/978-3-540-88875-8_12.

Zhao, W., Zeng, Q., Zheng, G., Yang, L., 2017. The resource allocation model for multi-process instances based on particle swarm optimization. Inf. Syst. Front. 19 (5), 1057–1066. http://dx.doi.org/10.1007/s10796-017-9743-5.