Schwemmer, Julia; Günsel, Christian; Kühn, Mathias; Schmidt, Thorsten

Article

Workforce rostering for decentrally controlled production systems: A simulation-based optimization framework using a genetic algorithm

Logistics Research

Provided in Cooperation with:
Bundesvereinigung Logistik (BVL) e.V., Bremen

Suggested Citation: Schwemmer, Julia; Günsel, Christian; Kühn, Mathias; Schmidt, Thorsten (2023) : Workforce rostering for decentrally controlled production systems: A simulation-based optimization framework using a genetic algorithm, Logistics Research, ISSN 1865-0368, Bundesvereinigung Logistik (BVL), Bremen, Vol. 16, Iss. 1, pp. 1-26, https://doi.org/10.23773/2023_8

This Version is available at:
https://hdl.handle.net/10419/297212

Logistics Research (2023) 16:8
DOI_10.23773/2023_8

Workforce Rostering for Decentrally Controlled Production Systems: A Simulation-based Optimization Framework using a Genetic Algorithm

J. Schwemmer, C. Günsel, M. Kühn and T. Schmidt

Received: 30 May 2022 / Accepted: 20 June 2023 / Published online: 19 July 2023 ©

ABSTRACT

Decentral production control plays a crucial role within the paradigm of Industry 4.0. Due to the fast and flexible decisions on allocation and sequencing required by this type of control, there is no baseline production schedule in advance. This creates a dilemma for efficient staff deployment – typically worker deployment times must be planned at least a few days ahead. To solve this dilemma, we present a simulation-based genetic algorithm, which creates a roster with flexible deployment intervals without a rigid shift pattern based on the production system and job load. In accordance with the zeitgeist and Industry 5.0, we include flexible working time and desired working hours of production workers. For evaluation of the method, we consider worker attendance costs, job delay costs and a cost penalty of work scheduled outside of desired working hours. We forecast the decisions of the decentralized production system by solving a job shop scheduling problem (JSP) extended by manual operations. Our algorithm iteratively uses reasonable solutions of the JSP as basis for roster optimization. With this integrated approach, it is possible to balance job delay costs against worker attendance costs as well as cost for deviation from desired working hours. To ensure compliance with working time legislation, we include appropriate repair operators in the genetic algorithm. We demonstrate the efficiency of our heuristic approach by comparison to rigid shift systems and the best of a large number of randomly created rosters.

KEYWORDS: Integrated workforce rostering and job shop scheduling problem · workforce requirement planning · decentral production control · Industry 4.0 · genetic algorithm with repair operators

This article is part of a focus collection on "Dynamics in Logistics – Models and Algorithms for Optimisation, Planning and Control."

Julia Schwemmer
Christian Günsel
Mathias Kühn
Thorsten Schmidt

Chair of Material Handling and Logistics Engineering, TU Dresden, Germany,

julia.schwemmer@tu-dresden.de
telephone: +49 351 463 33492
fax: +49 351 463 35499
www.tu-dresden.de/ing/maschinenwesen/itla/tl

1 INTRODUCTION

1.1 Workforce Scheduling Dilemma in Industry 4.0

The concept of Industry 4.0 brings many technical and organizational changes. This does not stop at human resource planning and scheduling or at the nature of employee tasks either, as we have already discussed in [1]. However, a large part of the research community agrees that humans will continue to play a central role in the smart factory [2, 3]. In this contribution, we particularly focus on machine-dominated manufacturing environments. We do not refer to a mainly manually dominated project manufacturing like complex assembly processes, as described, e.g., in [4]. Especially in machine-dominated production environments, the nature of the tasks of the production workers will change [2, 3]. Within cyber-physical production systems (CPPS) humans will mainly have coordinating, controlling and directing tasks [2, 3]. Within the upcoming trend Industry 5.0 [5], the human-centric aspect becomes a new core issue. The progressive automation and digitalization will make simple tasks obsolete that are usually characterized by continuous time-requirements. Consequently, steady deployment continuity of manual tasks during the production process will decrease.

The fourth industrial revolution will have impact on the organizational level of workforce scheduling. A basic element of Industry 4.0 is the decentralization of production control [6]. Thus, decisions on sequence and allocation of resources and orders will take place at lowest shop floor level. This will enable a high degree of flexibility and very rapid reactivity to process disturbances in production control. Decisions without long lead times will create a real-time control (as it is commonly called). There will be no (detailed) basic schedule for the production system. It will not be known (at least not in detail), which operation will be scheduled on which machine at what time. At this point, the dilemma of workforce scheduling in Industry 4.0 arises. In contrast to the production schedule, the staff roster has to be determined some days or weeks in advance so the workers have the opportunity for proper time management. In order to create an efficient roster by conventional methods, the information of the requirements from the production would have to be already available. However, with decentralized control, this information is not available due to the missing basic production schedule. This impedes efficient resource planning of the workforce.

In addition to technical and organizational changes (see also "Work 4.0" [7]), there is a transformation in the attitude of work. This tendency is sometimes also referred to as New Work [8]. Both trends influence each other. Work-life balance is one key aspect increasingly coming to the fore [7]. This means that the balance between working time and free time gains higher relevance to employees. The main focus is no longer on career and salary. In contrast, compatibility of work and family or hobbies increases in importance. Companies must adapt to the changing wishes of their employees in order to be able to recruit and retain skilled personnel. Especially in highly specialized areas, a scarcity of skilled workers cannot be ruled out in the future [2].

1.2 Scope and Solution Approach

In this work, we are not going to completely solve the dilemma of intersection "scheduling vs. planning" in decentrally controlled production systems. However, we take a first step for the rostering of the workforce ("planning"-side) to enable an effective "scheduling"-side.

We take advantage of flexible working hours and the rosters do not have to stick to fixed time grids of rigid shift systems. Flexible working hours may not only be beneficial from the employer's perspective to adjust capacity supply to a volatile demand but also from the employee's perspective to reach a better work-life balance when working at desired working hours. These requirements from both sides are not necessarily compatible, but we consider both and try to reach a compromise.

The core idea of our solution approach is a simulation-based optimization model. Utilizing the simulation model, we can evaluate proposed rosters and predict the behavior of the decentral production control for single roster proposals. We use discrete-event simulation. Utilizing a specifically adapted genetic algorithm as optimization model, we search for efficient roster proposals to improve the overall performance of the production system. Against the background of variable working hours, we encode solution proposals in chromosomes of variable length. Thus, we enable differing numbers of working intervals (deployment intervals) for the workers in the solution finding process. We have specifically adapted the genetic operators, which are messy crossover, simulated binary crossover and Gaussian mutation. In addition, we have two purpose-built operators which can change the chromosome length.

To deal with the fact that our problem is highly constrained due to working time legislation, we include a repair method consisting of five main repair operators. These are applied after the genetic operators and ensure that the legal requirements are met.

To achieve overall optimization of the production system, the objective function consists of three main terms: cost of rostered worker attendance, deviation from desired working hours from the worker's perspective and job delay. We evaluate the method by a comparison with classical two-shift and three-shift systems on the one hand and with the best of a large number of randomly generated rosters on the other hand.

As problem case, we use the class of job shop scheduling problems (JSP) with the specification of time-windows (release and due dates) as well as multi-mode (varying parallel production schedule schemes). With regard to the manual tasks, a real-world application field is, for example, the production of medical implants (examples of manual tasks: visual inspection, insertion and removal of parts or performing non-standard production steps). We chose the JSP because it models machine-dominated production environments where manual tasks may still appear. Alternative problem cases, underlying workforce rostering, could be Line Balancing Problems or Resource Constrained Project Scheduling Problems (RCPSP) (see, e.g., [9]). However, it seems to us that these alternatives do not balance machine tasks and human tasks as well as the JSP.

We do not attempt a novel optimization technique for the JSP. In order to take up the planning background of Industry 4.0, we solve the JSP by modelling a decentralized production control. The JSP solution is intended as a reasonable forecast of the decisions made by the decentralized production control.

As for the solution method of our integrated rostering and scheduling problem, we are interested in a heuristic approach – in our case a GA – given the NP-hardness of the problem [10].

In Sect. 2, we give a rough overview of the state of the art that we have already presented in more detail in [11]. In Sect. 3, we precisely define our considered problem. Section 4 deals with a proposed solution

method which is based on the genetic algorithm adapted for the problem stated. In Sect. 5, we present some results of our optimization and simulation runs to evaluate the proposed algorithm. The paper ends with a short conclusion and planned extensions for the algorithm in Sect. 6.

2 STATE OF THE ART

The human-centric aspect, to which our method of workforce rostering refers, is currently gaining importance and public interest. The new trend "Industry 5.0" [5] highlights exactly this human-centric perspective. A more general view of the topic of social sustainability in production planning is presented in the literature review of Trost et al. [12].

We have conducted a detailed literature review with focus on quantitative solution methods for roster optimization in [11], covering the following aspects:

- parallel optimization of roster and machine plus workforce scheduling (i.e., job scheduling, staff rostering and staff assignment – see also [9]),
- time-flexible working models,
- consideration of working hours preferred by workers,
- decentralized production control with short, sparse manual production tasks.

Since publication of [11], there have not yet been any significant changes in the state of the art. So, we would like to refer to our paper [11] for more detailed information on the literature status. We give a short summary of the state of the art in this publication:

Historical Development

Flexible working time arrangements are not a new concept. Already at the turn of 2000, some concepts based on time accounts were introduced in the literature (see, e.g., [13]). These have primarily aimed at the adjustment of capacities to market fluctuations and thus realization of employer-side advantages. There was usually no inclusion of employee-side desired working hours. The workforce scheduling process was usually a second level optimization, optimized after finishing the machine sequencing and allocation (see, e.g., [14]). Therefore, it will not meet the described future requirements of Sect. 1.1.

In 2011, Kagermann, Lukas and Wahlster [15] presented the concept of the fourth industrial revolution at the Hannover Fair the first time. From 2012 onwards, the German government also encouraged it (see, e.g., [6]) and the number of publications dealing with Industry 4.0 began to rise. Accordingly, corresponding concepts based on decentralized production control became more important. There are several research projects that deal with the technical collaboration of human and machine (e.g. see [16]). However, the organizational aspects of flexible shifting workforce in CPPS have hardly been considered [11]. As the decentrally controlled production system is one main issue in our research area, we have limited the literature review to the application area of manufacturing and quantitative contributions since 2011.

From the employees' point of view, there is a trend with strong influence on the research topic, too. For several years, work-life balance has been gaining in importance, especially based on flexible working hours that can be influenced by the individual workers themselves. (see, e.g., [7, 17]).

Related Problem Classes

To get a general overview on the state of the art of the topic of staff scheduling and rostering, we recommend the three literature reviews [18–20]. Usually, the problem classes considered in these reviews only cover the staffing optimization component without taking into account the scheduling side of machine and job optimization [11]. For example, looking specifically at our described dilemma, the class of "Employee-Timetabling" (see, e.g., [21–23]) is one of the first related problem classes. However, this class commonly only addresses human resource planning but not the machine shop problem [11]. Considering the transformation in the attitude of work as described at the end of Sect. 1.1, there is hardly any literature: Firstly, there are hardly any methods to roster workforce without rigid shift grid on the basis of time flexible working models [11]. Secondly, there are even less publications that include the aspect of employee-sided desired working [11].

So, on the one hand, there are publications dealing with workforce rostering [11]. On the other hand, there are publications dealing with the scheduling processes of machine operations and job sequences [11]. These are, for example, Job Shop Scheduling Problems (JSP/JSSP), Flow Shop Scheduling Problem (FSP/FSSP), Flexible JSP (FJSP/FJSSP) or Line Balancing Problems [11]. Some of these problem classes have extensions where also human operations are to be scheduled (see, e.g., [24–29]) but they do not take into account the topic of workforce rostering in general or at least not at the same optimization level as the machine scheduling problem [11]. A class of scheduling problems constrained by machine capacity and by human capacity is commonly called dual resource constrained (DRC), where the term DRC is mainly used in the context of job shop scheduling problems. In DRC problems, the working hours are usually taken as a given and the rostering process is mostly not a subject of investigation [11]. Some of the DRC problems of the recent years are written in the context of Industry 4.0 (see, e.g., [24, 30]) and some do not take up the topic of Industry 4.0 (see, e.g., [25, 31]). Generally, in the field of workforce rostering or DRC scheduling problems, literature considering the background of a decentrally controlled production is rare [11].

Taking further facets of our considered issue into account, like the changed time requirements of the manual tasks, there are only very few publications that

model problem instances with the sort of sparse and short manual activities as described in Sect. 1.1 [11].

There are some contributions that do not assign their proposed model to a commonly known problem class. The phenomenon described above can also be observed here: either the publications are attributed to a pure personnel problem (see, e.g., [14]) or, if a machine problem is included, the rostering component is not taken into account (see, e.g., [32–35]).

There are some publications that include the problem classes of job scheduling, staff rostering and staff assignment in an integrated problem class (see, e.g., [9, 10, 36, 37]). However, the problem conditions of these integrated problems differ from our requirements: For example, [10] consider a lexicographic objective function giving an order of the various objective functions. This in turn does not imply an optimization of the different problem classes (job + staff scheduling and staff rostering) on an equal optimization level. Mostly these integrated approaches do not include flexible shift planning but shifts have to be of the same length (see, e.g., [10, 36]). In general, we could not find any integrated staff rostering and scheduling problem that takes desired working time conditions from the employee's side into account.

To summarize, to the best of our knowledge, there are only publications dealing with related or reduced problems but there is no method handling exactly the dilemma described in Sect. 1.1 or more specifically within the scope of the research gap in the next paragraph. Furthermore, we did not find benchmark instances or problem model formulations for the precise research question we are interested in.

Research Gap

To summarize, the research gap our paper aims at (see also [11]) is characterized by a combination of the following requirements, which are to the best or our knowledge not sufficiently addressed by current research:

- The production planning and control of machines, worker assignment and jobs as well as the roster generation for employees are merged on the same level of importance. There should be just one optimization level and no subsequent, "second-level" optimization for one of the components,
- Both workers and machines are considered as separate but equally important and limited resources,
- The planning of deployment times is completely independent of shift grids,
- Working time preferences from production employees are included, and
- The background of a decentrally controlled production system with changed processing time requirements of manual production tasks is considered.

To give another example, the method of the research project "KapaflexCy" [38] requests available workers via a "shift doodle" app for additional shifts or shift extensions at short notice but only after the demand planning at machine level has been completed. Furthermore, the rostering system is based on a fixed rigid shift pattern.

3 PROBLEM STATEMENT

The aim of our research project is to find a compromise between the predictability of workers' working hours with sufficient lead time and the flexibility of decentralized control.

The goal of our research is to determine an optimal workforce roster relative to a given job shop problem instance. We strive for an overall optimization that takes into account cost for worker attendance (labor cost) as well as cost for an understaffed production process. For the latter, we will use the target value of job delay that is caused by periods when less workers are available than required to finish the job in time. Our third criterion for optimization is to minimize the cost for the deviation of working time preferences of workers from actual worker deployment in the roster. Thus, we take into account the social tendency of the changing attitude of work.

The concrete aim of this publication is only to create cost-efficient worker deployment rosters, not to determine schedules with sequence and allocation decisions for the underlying job shop problem. Fixing all dispatching decisions for the operative level of the production system would clash with the decentral production control, which makes dispatching decisions when they are due and does not need a fixed baseline schedule. In these terms, the deployment rosters only determine attendance times. The exact assignment of workers to tasks should finally take place in the short term by the decentralized production control. The operative level is still in the decision-making power of the decentralized control.

However, in order to achieve an equivalent inclusion of the rostering and the machine level (as described in Sect. 1.1 and 2), it is essential to deal with the machine shop as well. The production runs carried out must also be included in the evaluation – even if they are not part of the solution – but they can be viewed as a forecast of the operative decisions made on the shop floor. For the detailed relationship between the both components, please see Sect. 3.1.

3.1 Relation of Workforce Roster to Job Shop Scheduling Problem (JSP)

A workforce roster defines the attendance times (deployment intervals) of each staff member. Only if a worker is attendant in the production system, a manual task (part of the JSP) can be processed. In this way, the workforce roster (planning in advance) is covering

the demand for the resource human of the production system (tasks included in the JSP on an operative level without lead time).

To determine costs for a workforce roster and the instance of a job shop scheduling problem (JSP), the operations of the JSP instance must include manual activities which require the attendance of workers. A "solution" to this extended JSP (which in the sense of the problem description is not a solution of the optimization problem but only a forecast of a possible production run) consists in a concrete schedule in which the start and end of each job, operation, and manual activity is mapped onto the time axis. Any such forecast will take the availability of workers based on the given workforce roster into account and thus represent a valid production run, which fulfills the constraints of a JSP (see Sect. 3.2). In this way, the concrete timing of the start and end of jobs determined by the forecast will give rise to delay costs if a job ends after its due date.

The attendance costs result from both perspectives – the roster (planned attendance time) and the forecasted production run (unplanned attendance time but scheduled in the forecast) – as our model allows to work extra hours to complete already started production tasks.

In contrast to the delay costs and attendance costs, which depend on the JSP, the costs of deviation from preferred hours can be calculated only on the basis of the workforce roster itself.

The forecasted production run estimates how the rostered working hours affect the processing of manual tasks (e.g., are adequate worker available) and thus the processing of jobs (e.g., waiting times of jobs or blocking times of machines). On this basis, the workforce roster can be prepared in advance.

We do not attempt a novel optimization technique for the JSP. To achieve our aim of roster optimization, however, we need to assume and implement some solution of the JSP – as forecast of decentral production control – which does not need to be optimal but just reasonable.

The flexibility on the side of decentral production control is to be modeled by using a multi-mode problem, in which there are alternative processing options for each production step. The processing options can vary in processing times and required resources (machine as well as human) and can differ for every job operation. Since resources are being shared, the scheduling of one job depends on the scheduling of other jobs. Accordingly, at almost every decision point in time, there are different parallel production schemes possible. Selecting one of them is the task of the decentral production control. Thus, the decentral production control determines allocation of resources and sequence of job operations, which will become the forecast of the production run.

The scheduling logic of decentralized production control is typically either agent-based or priority rule-based. In this problem formulation, we consider the decentralized production control as given and model it on the basis of priority rules – "first-in-first-out" followed by "shortest processing time".

In the following subsections, we give more details of the extended JSP instance, the workforce roster, and the roster optimization problem.

3.2 Models for Extended JSP and Workforce Roster

In the following, we specify the problem in a semi-formal way. The following equations are an excerpt of the necessary mathematic modeling:

3.2.1 Model of the Extended JSP

As starting point, we use the class of Job Shop Scheduling Problems (JSP) with specifications of multi-mode and time windows (MJSPTW). There are machines (or workstations) $m \in \{1, \dots, |M|\}$ and jobs $j \in \{1, \dots, |J|\}$. Each job j has a set of operations $o_j \in \{1, \dots, |O_j|\}$. There is a fixed sequence for the operations (predecessor/successor). Each operation o_j has a set of modes $q_{jo} \in \{1, \dots, |Q_{jo}|\}$. The modes in Q_{jo} represent alternatives of operation o_j, of which only one must be executed. A mode q_{jo} determines the required machine and the duration $\tau(q_{jo})$ of the machine usage. Each machine m can process only one operation o_j at a time.

Constraints for the JSP

A valid solution of the JSP is characterized by the complete processing of all jobs, where for each operation o_j of a job j, exactly one operation mode q_{jo} has to be selected. These selected modes are determined by a start and end time such that $\text{end}(q_{jo}) - \text{start}(q_{jo}) \geq \text{duration}(q_{jo})$. The start of the mode $q_{jo} + 1$ selected for the next operation $o_j + 1$ of job j must be after the end of the mode q_{jo}. In other words, modes selected for operations of the same job must not overlap. Each mode q_{jo} uses a concrete machine as defined in the JSP instance. The same machine may only be used by two different selected modes if the modes do not overlap in terms of their start and end.

Extended JSP

We extend this classic JSP by two aspects:

Firstly, the standard JSP is extended by assigning to each job j a planned release date α_j^{plan} and a required due date ω_j^{plan}, which will be used to define delay cost.

Secondly, for the workforce planning component of our use case, we have to extend the MJSPTW with manual activities a_{joq} (for a concrete example see Sect. 5.1). We do not distinguish between different qualifications of the workers; so, all workers are interchangeable (homogeneous qualification). The idea is, that each worker $w \in \{1, \dots, |W|\}$ can perform every manual activity a_{joq}. However, a worker w can perform only one manual activity a_{joq} at the same time. Thus, if an operation o_j has already started on a machine m

and then a worker is required due to a manual activity a_{joq}, the operation o_j must be suspended until a worker is available. We will formulate this requirement as a constraint when connecting the extended JSP model with the roster model in Subsect. 3.2.3.

We model the problem in such a way that manual activities a_{joq} do not occur alone, but a machine operation o_j is always used as a basis. The machine m remains occupied during the complete operation o_j until the operation is finished. The idea is that a manual activity always needs an operating machine m (or some workstation m). Manual activities a_{joq} can be shorter but not longer than their machine operation o_j. By assigning rather short manual activities to a rather small subset of machine operations, we can model a situation as described in Sect. 1.1. Due to the extension of machine operations with manual activities, machine operations o_j are no longer the smallest unit in the scheduling process – contrary to a standard JSP.

Constraint for the Extended JSP

A solution of the extended JSP assigns a start $\text{start}(a_{joq})$ and end $\text{end}(a_{joq})$ to all manual activities a_{joq} that belong to the selected operation modes such that $\text{end}(a_{joq}) - \text{start}(a_{joq}) \geq \text{duration}(a_{joq})$.

3.2.2 Model of the Workforce Roster

The concrete aim of the algorithm described in this publication is to create cost-efficient worker deployment rosters. We explicitly do not talk about schedules as solutions, since we only want to determine attendance times. The exact assignment of workers to tasks should finally take place in the short term through decentralized control and is not within the scope of our problem.

For further considerations, we would like to define and distinguish the following terms:

- ***Deployment interval***: A period of time with a defined starting and end point. In this interval, a worker is present and ready to work in the company. The list of all deployment intervals thus forms the working hours of an employee. Technical and personal disruptions times (contingency allowance) may occur during the interval, but designated breaks are not included.
- ***Break***: A break separates *deployment intervals* (e.g., lunch break). Breaks have a certain minimum duration, but are not as long as rest periods. In practice, breaks are commonly between 0.5 – 2 hours long.
- ***Rest period***: A several hours long recovery time between shifts in which the employee has time for his/her personal needs (e.g., sleep time) and interests (e.g., family and hobbies).

  Shift: A shift consists of one or several *deployment intervals* and, if required, *breaks*. Two shifts are separated by a rest period, where the *rest period* is not part of the shift time.

A roster consists of a list of deployment intervals for each worker w. The roster for worker w is defined as:

$$D_w^{plan} = \{d_{w1}^{plan}, \dots, d_{w|D_w|}^{plan}\} \qquad \forall w \in W \quad (1)$$

where d_{wi}^{plan} is a deployment interval of worker w:

$$d_{wi}^{plan} = \left[t_{s_{wi}}, t_{e_{wi}}\right] \quad \forall w \in W; i \in \{1, \dots, \left|D_w^{plan}\right|\} \quad (2)$$

where $t_{s_{wi}}$ is the start[1] of a deployment interval d_{wi}^{plan} and $t_{e_{wi}}$ is the end of the deployment interval d_{wi}^{plan}. The roster for all workers is simply the set $\{ D_w^{plan} \mid w \in W\}$.

3.2.3 Model Connection of the Extended JSP and the Workforce Roster Problem

The following constraint connects the extended JSP with the workforce roster: For each point in time t, a manual activity a_{joq} may only start at t, if there are more workers deployed at t than there are active manual activities at t.

In this formulation, the number of workers deployed at t is the number of workers w for which there is a deployment interval d_{wi}^{plan} such that $t \in d_{wi}^{plan}$. The start and end of a manual activity a_{joq} is fixed by the solution of the extended JSP. Thus, it is well defined at which times t a manual activity is active, namely if $\text{start}(a_{joq}) \leq t < \text{end}(a_{joq})$.

Thus, it is allowed to pass manual activities a_{joq} between workers, since the constraint does not require that individual workers are assigned to activities.

Note that this constraint enforces that workforce rosters must have sufficiently many deployment intervals of sufficient lengths to allow the required assignment of start and end to all manual activities. It thus guarantees that all manual activities are completed.

The forecasted solution of the extended JSP problem under the stated conditions implicitly also fixes the start and end time of each job that we refer to as α_j^{eff} and ω_j^{eff}, respectively. On the basis of ω_j^{eff}, we can define job delay costs by comparing it with ω_j^{plan}.

3.2.4 Preferred Working Hours

The labor flexibility on the worker side is to be implemented as follows: Workers can indicate preferred working hours that are not bound to any fixed shift pattern, but are simply the times at which they would like to work. These working hour requests are taken into account when creating the rosters but they are not obligatory. This means that the rosters do not necessarily cover the workers' wishes exactly. Rostered working hours outside the desired working time window add costs to the target function value (e.g.,

[1] Note that the start time $t_{s_{wi}}$ and end time $t_{e_{wi}}$ are integers, which refer to simulation time units used by our discrete-event simulation of job execution in a production system. In order to calculate hour-based costs, one has to interpret simulation time units in real time (e.g., 1 simulation time unit = 10 min, see Sect. 5.1)

in transfer to the real world, it can be interpreted as a bonus payment for the worker).

We will denote the desired deployment intervals for each worker by

$$D_w^{wish} = \{d_{w1}^{wish}, \dots, d_{w|D_w|}^{wish}\}$$

3.2.5 Overview of Time Dependent Variables

According to the constraints, a worker can only perform a manual activity a_{joq} if the start of the activity a_{joq} is during his/her rostered working hours D_w^{plan}. So, a worker can start to work on a task during the rostered working hours D_w^{plan}. However, we allow that if the worker is not able to finish the already started activity a_{joq} within the rostered working hours D_w^{plan} and there is no free worker to hand over the task to, the worker will exceed his/her planned working hours and finish the task. Therefore, the rostered working hours D_w^{plan} and the effective working hours D_w^{eff} realized in the simulation can differ. So, there might be workers effectively deployed outside the times fixed for them in the roster.

For this reason, we have to distinguish two to three versions of time dependent variables (see Table 1): plan, effective and, if applicable, wish. For example, each worker w has preferred working hours D_w^{wish} but he/she has no preferred release date α_j^{wish} for job j.

3.3 Decision Variables and Objective Function of the Optimization Problem

Decision Variables

The decision variables are the rostered attendance times of the workers $D_w^{plan} = \left\{d_{w1}^{plan}, \dots, d_{w|D_w|}^{plan}\right\} = \{\left[t_{s_{w1}}^{plan}, t_{e_{w1}}^{plan}\right], \dots, \left[t_{s_{w|D_w|}}^{plan}, t_{e_{w|D_w|}}^{plan}\right]\}$. Due to the flexible setting of the working hours and the flexible time horizon, there is no explicit restriction to the number of deployment intervals d_{wi}^{plan}, but other constraints (e.g., breaks and rest periods of legal restrictions) must be kept and they indirectly influence the number of feasible deployment intervals. For example, a minimum break length of one hour will limit the number of deployment intervals per day to <24. A mandatory rest period of 11 hours per day will further reduce this number. However, we do not assume a fixed number of deployment intervals, either per day or overall, within the time horizon. This means that there is no fixed number of decision variables, even for a given problem instance.

Objective Function

We measure all components of the objective function as cost (e.g., represented in €) to create better comparability between them. The total costs $c_{objective}$ are to be minimized. The objective function consists of three parts: The costs for worker attendance $c_{attendance}$ (labor cost), costs for working hours deviating from the desired working hours $c_{undesired}$ and the costs for job delay c_{delay}.

$$c_{objective} = c_{attendance} + c_{undesired} + c_{delay} \rightarrow \min \quad (3)$$

A roster with minimal costs thus provides a compromise solution between the worker's wishes, the requirements from the production system, and the costs from the worker deployments. In a real production scenario, fixing working hours in advance using such a minimal-cost roster creates planning reliability for workers while still optimizing for low labor cost and delay cost.

Cost of Worker Attendance $c_{attendance}$

The cost of attendance (labor cost) is calculated as follows:

$$c_{attendance} = \sum_{w=1}^{|W|} \sum_{i=1}^{|D_w|} \mathrm{Day}\left(d_{wi}^{eff}\right) * a_w + \mathrm{Night}\left(d_{wi}^{eff}\right) * a_w * \left(1 + b_{night}\right) \quad (4)$$

Here, $\mathrm{Day}(d)$ is the partial duration of deployment interval d that falls in the day time (e.g., 6:00 a.m. to 11:00 p.m.) and $\mathrm{Night}(d)$ is the partial duration of d

Table 1: The notation for the different versions of time dependent variables

	Plan	Effective / simulated	Wish (from the worker's perspective)
Set of all worker deployment intervals	$D_w^{plan} = \{d_{w1}^{plan}, \dots, d_{w\|D_w\|}^{plan}\}$	$D_w^{eff} = \{d_{w1}^{eff}, \dots, d_{w\|D_w\|}^{eff}\}$	$D_w^{wish} = \{d_{w1}^{wish}, \dots, d_{w\|D_w\|}^{wish}\}$
Single worker deployment interval	$d_{wi}^{plan} = \left[t_{s_{wi}}^{plan}, t_{e_{wi}}^{plan}\right]$	$d_{wi}^{eff} = \left[t_{s_{wi}}^{eff}, t_{e_{wi}}^{eff}\right]$	$d_{wi}^{wish} = \left[t_{s_{wi}}^{wish}, t_{e_{wi}}^{wish}\right]$
Job release date	α_j^{plan}	α_j^{eff}	–
Job due date	ω_j^{plan}	ω_j^{eff}	–

that falls in the night time (e.g., 11:00 p.m. to 6:00 a.m.). Both are measured in hours.

a_w stands for the labour costs per hour of worker w. Note that, in the current stage of development, all a_w are equal since workers do not have different qualifications.

b_{night} is the night work surcharge, e.g., 25% = 0.25.

Cost of Desired Working Hours $c_{undesired}$

$$c_{undesired} = \sum_{w=1}^{|W|} \text{Difference}(D_w^{plan}, D_w^{wish}) * a_w * b_{wish} \quad (5)$$

where $\text{Difference}(D_w^{plan}, D_w^{wish})$ is the overall amount of time in the intervals from D_w^{plan} that is **outside** all intervals from D_w^{wish}. This amount of time, which lies outside of the desired working hours, is measured in hours. b_{wish} is a percentage bonus surcharge factor applied to it.

Cost of Job Delay c_{delay}

If jobs are not completed by the due date ω_j^{plan}, a penalty is paid, which is determined by a cost rate per hour r_j (e.g., contract penalty).

$$c_{delay} = \sum_{j=1}^{|J|} \max\left(0, \omega_j^{eff} - \omega_j^{plan}\right) * r_j \quad (6)$$

Constraints due to Work-Hour Legislation

In a staff rostering problem such as ours, numerous restrictions need to be taken into account that exist to protect workers. This makes the problem more complex (highly constrained) because the search space becomes more fragmented. We derive the main constraints from the German Working Time Act (e.g., maximal work load per day or minimal break length). The applied upper and lower bounds are in Table 2. The instantiation of constraints in the later test series will be on the specifications of this German law. However, these parameters can be set according to individual needs (e.g., country-specific or company-specific requirements).

Once a worker has worked a certain amount of time (e.g., see $u_{interval}$), a break should be taken. There are also specified minimum times for breaks. In the German Working Time Act this depends on the daily working time. For example, this is at least half an hour for more than one day's work between 6 and 9 hours and a 45-minute break for longer working hours. To simplify matters, we set a minimum break value l_{break} of 1h, as this means that all specifications are met using a single value. Smaller breaks (e.g., personal disruptions times as contingency allowance) should not be shown in an official work schedule of the worker (take in mind the level of workforce planning of HR). Smaller breaks fall into the personal distribution time of the contingency allowance (see definition in Sec. 1).

We added $l_{interval}$ to avoid scattering of working time across many short deployment intervals and breaks but this is not a legislative rule. The aim here is to avoid creating overly fragmented intervals of work that are impractical in real-world applications or generally in human resource planning. For example, a half-hour interval of working time is created, separated from breaks or even standing alone for that

Table 2: Applied thresholds for the arrangement of working hours

		used configuration
l_{break}	Minimal break duration (within a shift)	1h **
l_{rest}	Minimal duration of a rest period between shifts	11h
$l_{interval}$	Minimal duration of a deployment interval	1h *
$u_{interval}$	Maximal duration of a deployment interval	6 h
u_{day}	Maximal number of working hours per day	8 h
u_{shift}	Maximal number of working hours per shift	8 h *
$u_{shiftspan}$	Maximal length of a shift (start of first deployment interval until end of last deployment interval in the shift)	13 h *

* *not directly given in the German Working Time Act*

** *simplified*

day. The employer will undoubtedly agree with the employee's view that this is not an acceptable work schedule in which the value-added times are sufficiently proportionate to the employee's organizational efforts (traveling to the company, preparing protective clothing or special tools if necessary). In addition, the day of the employee is interrupted in an inappropriate way. Due to the employer's duty of care towards his employees, the minimal duration for a working interval $l_{interval}$ is set.

The maximum time load of work within a shift u_{shift} is also not directly given in the German Working Time Act. However, it also appears to us as an important parameter and we set it to 8 h. We have transferred this restriction from the legal condition of the maximum working time of 8 h within one day. The difference between the maximum daily working time and the maximum shift working time is that the first one refers to the 24-hours clock-grid, but the latter can take place across days (e.g., classic night-shift times from 10:00 p.m. to 6:00 a.m.).

In addition, the maximum shift length $u_{shiftspan}$ (containing deployment intervals and breaks of one specific shift) is a parameter that is not directly given either. However, the modeling point of view shows its necessity. We calculate it from the length of a day (24 h) minus minimal rest period (11 h): 13 h. That means, after a completely exhausted shift length (distinguished from the working time within a shift) and the minimum rest period, the worker can start his working time at same time on the next day. However, the planned shift length can be shorter and/or the planned rest period can be longer, so the shift-grid is not based on the day-grid. Shift times may switch over the daily grid.

On this step of development of our approach, there is not yet any special focus on consideration of psychological and work science aspects; and there is no consideration of exactly which tasks are performed (e.g., parts replacement, visual inspections).

4 SIMULATION-BASED OPTIMIZATION VIA GENETIC ALGORITHM

To solve the problem described in the previous section, we use a simulation-based optimization model. With regard to the large size of industrial rostering problems as well as the size of decision variable search space through a time-flexible rostering scheme, a simulation-based approach as opposed to an approach using mathematical programming seems appropriate for us. As stated in [10], both problem classes (staff rostering and job shop scheduling) are each NP-hard optimization problems and an integrated form of both will not reduce the problem complexity. With increasing computational power, also increasing complex problems can be solved mathematically exactly (within an acceptable period of time), e.g., via a mixed integer linear programming model (for further information on this discussion, please see, e.g., [39, 40]). Based on currently available computational power and considering our problem complexity, we decided to use a heuristic method in this case.

A general schema for our optimization model is shown in Fig. 1.

As optimizer, we have adapted a genetic algorithm (GA). A GA is a stochastic optimization method, working as a heuristic that searches for a global solution by the imitation of an evolutionary process (for more information see, e.g., [41]).

Our GA uses a discrete-event simulation model for the evaluation of the objective function. For software implementation, we use the programming language Python (version 3.7) as well as the package DEAP (version 1.3.1) [42] for the GA and the package SimPy (version 4.0.1) [43] for the simulation.

Fig. 2 shows an outline of our genetic algorithm, which implements the general schema of Fig. 1.

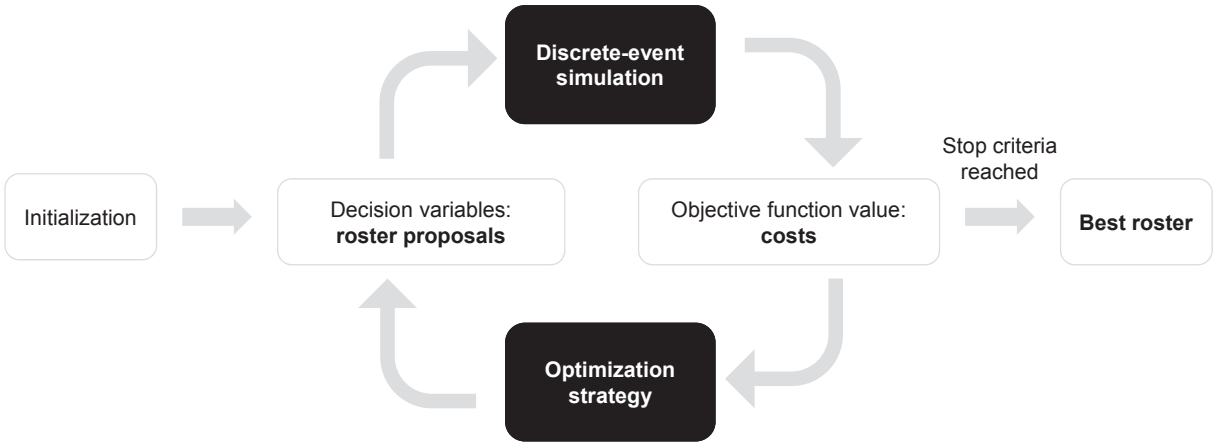

Fig. 1: General schema of our simulation-based optimization model

Algorithm 1: GA optimization model

1: initialization: generate start population
2: **while** stop criteria are false **do**
3: evaluate fitness of each individual
4: apply tournament selection
5: apply crossover
6: apply mutation
7: apply repair mechanism
8: **end while**
9: **return** best individual (roster)

Algorithm 1a: Discrete-event simulation

3.1: solve extended JSP for each individual (roster)
3.2: calculate objective function (cost) for each individual

Algorithm 1b: Repair mechanism

7.1: apply ARO 1 + ARO 2
7.2: apply MRO 1 + ARO 2
7.3: apply MRO 2 + ARO 2
7.4: apply MRO 3 + ARO 2
7.5: apply MRO 4 + ARO 2
7.6: apply MRO 5 + ARO 2

ARO… auxiliary repair operator
MRO… main repair operator

Fig. 2: Pseudocode of our developed algorithm (the repair mechanism is explained in Sect. 4.6)

Components of Algorithm 1 – GA Optimization Model

- **Generate start population:** Our genetic algorithm starts by creating a start population of individuals each representing a roster solution. In the start population each individual has two random deployment intervals per worker and per day of the planning horizon. See Sect. 4.1 for details.
- **Evaluate fitness of each individual (Algorithm 1a):** The algorithm simulates the fixed extended JSP problem for each individual (roster) and generates a solution forecast. It calculates each individual's fitness value on the basis of the cost value for the respective solution forecast as defined in Sect. 3.3. See Sect. 4.3 for a detailed account.
- **Apply tournament selection:** The algorithm selects the best individuals to base the next generation on, see Sect. 4.4.
- **Apply crossover and mutation (genetic operators):** In this step the next generation is created by applying crossover and mutation operators to the individuals selected in the previous step. We give a detailed explanation in Sect. 4.5.
- **Apply repair mechanism:** This step ensures that all individuals of the next generation created in the step before comply with working time legislation, e.g., the length of breaks. Our repair algorithm attempts to change the individual as little as possible in order to keep good solutions. See Sect. 4.6 for a detailed description of both auxiliary and main repair operators.
- **Return best individual:** The algorithm keeps track of the individual with the best fitness value ever encountered in any generation. This individual is returned and represents the optimized roster.

Before we describe the steps of the algorithm in detail, we need to define the structure of chromosomes that encode the individuals and roster.

4.1 Solution Encoding and Fitness Function: Chromosomes of Individuals

As specified in our problem statement, it is not the task to find the solution with the shortest production time, but with the lowest cost. The optimization is done by the decision variables that form the deployment intervals of the worker roster. The list of all deployment intervals of all workers makes up the vector of and for the chromosome representation. There are two main arguments of not working with a fixed length of this decision vector. On the one hand, there is the flexibility of how many working intervals each day are rostered. There can be none, one, or more than one deployment interval in a worker's day. On the other hand, there is the flexibility of how long the required time horizon of the best solution is. The length of the required roster depends strongly on the contents of the roster. Therefore, there is no fixed time horizon in advance. In summary, it is neither fixed how many deployment intervals there are per day nor how many days it will take to finish all jobs and therefore there is no fixed number of decision variables.

Note that it is important that the chromosomes are specific for each worker and do not just encode the

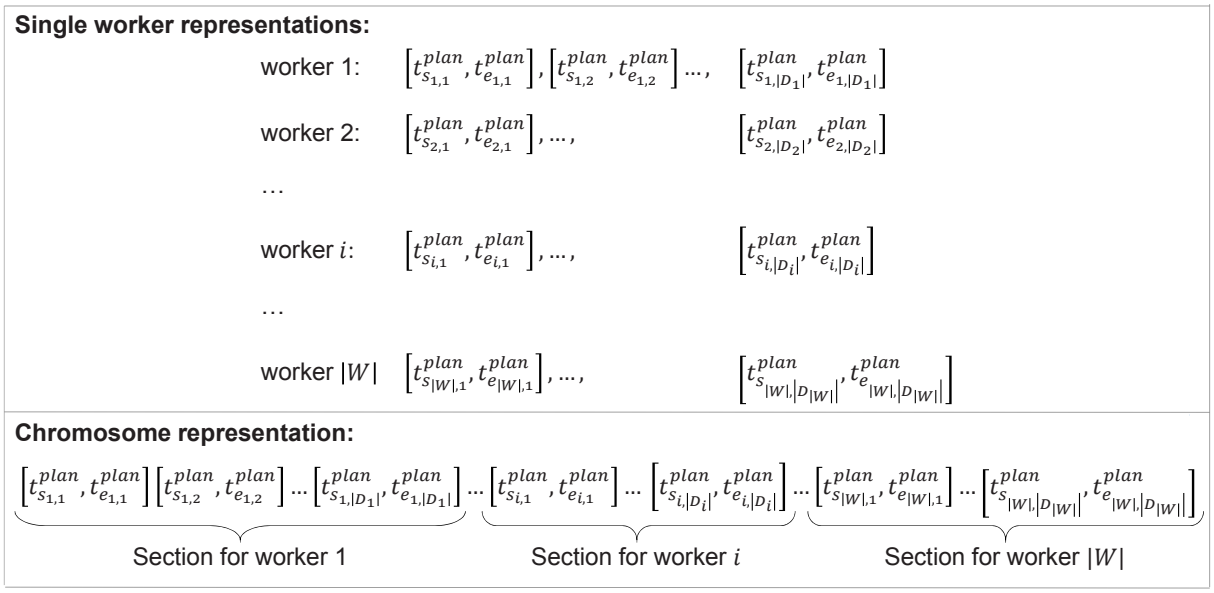

Fig. 3: Chromosome representation

number of workers deployed at certain times. Although our discrete-event simulation only uses the number of available workers (see Sect. 4.3), we need to make sure that the working time legislation (breaks, length of shifts, etc.) is complied with for each individual worker.

For the solution coding within the GA, the roster of all workers with all deployment intervals are in one chromosome (see Fig. 3). Accordingly, the chromosome consists of several sections: one for each worker. The deployment intervals for a worker are always sorted ascendingly by the start values within the respective worker section. All values are integers due to the problem formulation (discrete-event simulation). In this work, we use the terms *individual* and *chromosome* as synonyms.

The structure of the chromosomes and their variable length have a strong impact on the further process of the genetic algorithm. In the following, if we are referring to the variable chromosome length, we will mean the number of deployment intervals within a chromosome and not the time horizon.

Because of the unknown time horizon, it is very likely that the suggested solutions of the GA are too long or too short, considered in time. If the roster is longer than required, unnecessary costs arise as workers are scheduled without any tasks to do. However, this case is very easy to fix. During the evaluation process, the proposed roster is cut by the time step at which all jobs are finished. We shorten the roster only during the evaluation but not during further inheritance in the GA. Within the inheritance, it is easier to handle the time horizon if the rosters are longer than needed. In the other case, the roster ends too early. Then, the processing of orders cannot be completed because of a lack of workers. In this case, we use a penalty function to train the GA to avoid too short rosters. If the roster ends and not all jobs have already finished, all workers with their complete cost rates and an additional penalty are scheduled from the end of the roster until the jobs have been completed. During the development of the framework, we have observed that this is generally sufficient to prevent too short rosters.

Accordingly, the fitness function of the individuals is not only the objective function but combines the objective function value with a penalty for a too short roster $c^{penalty}_{afterschedule}$.

Additionally, the fitness function includes an extra penalty for unplanned working hours $c^{penalty}_{unplanned}$ (when workers finish an already started task beyond the end of their rostered working hours). This means that the rostered deployment interval was too short. We would like to avoid this case as far as possible but we handle this as a soft constraint.

In the following, we will refer to the value of the fitness function as $c_{fitness}$:

$$c_{fitness} = c_{objective} + c_{penalty} \rightarrow \min$$

$$\text{with } c_{penalty} = c^{penalty}_{afterschedule} + c^{penalty}_{unplanned} \tag{7}$$

4.2 Start Population and Stop Criteria

The generation of the start population of the GA is random-based. This requires a rough estimate of how long the time horizon (1 day, 1 week, 1 month etc.) of consideration is. Of course, the exact length for the best solution (best roster to be searched), does not need to be known. Instead, the length is adjusted during the optimization process. However, a roughly known time horizon significantly simplifies the search for the GA. We recommend to specify a period that is slightly too long to avoid the penalty of too short rosters.

The start individuals are randomly generated for the estimated period. For every worker, a fixed number of deployment intervals is generated, which is calculated from the length of the estimated time horizon and a parameter to be set for the initial number of intervals per day. We set an initial number of two working deployment intervals per day (e.g., one deployment interval before the lunch break and one afterwards) for our experiments. The start and duration of each interval are created randomly.

As an example, for an instance with a workload of 5 days, we initially generate 10 deployment intervals per worker. That means, with 5 workers, we will generate 50 deployment intervals. In a last step, we apply the repair algorithm to guarantee feasible deployments.

As stop criteria, we implemented two options. On the one hand, there is a maximum number of generations. On the other hand, the GA monitors the change in the value of the objective function. If there is no more improvement over a configurable number of evaluated generations, then the process stops.

4.3 Fitness-Evaluation of Each Individual: The Discrete-Event Simulation Model

While the extended JSP instance remains fixed throughout the GA, workforce rosters are represented by the evolving individuals. For each fitness evaluation of a workforce roster, the algorithm performs a full simulation of the extended JSP with respect to that roster. The roster is an input variable for the simulation; together with the jobs to be processed, the dispatching decisions are made on the basis of a given decentralized control. The decentralized control is not a parameter to be optimized in our problem case. As described in Sect. 3, it simply consists of priority rules: The main rule is "first-in-first-out" (FIFO), followed by the rule "shortest processing time" (SPT). With different job processing options on different machines (different modes), the job is allowed to virtually queue up at the same time on all possible machines. Only the mode that can be scheduled first is really processed, the other options are cancelled. For the resulting forecasted schedule, the algorithm calculates the costs defined in Sect. 3.3.

Our discrete-event simulation defines events as start and end of processes as well as requesting, granting and releasing of resources:

Machine Resource: For each machine $m \in \{1, \ldots, |M|\}$ there is a *machine resource* capable of carrying out one operation at a time. Each machine resource has a first-in-first-out (FIFO) queue for handling requests.

Workforce Resource: There is a single *workforce resource* with capacity $|W|$, i.e., capable of carrying out as many manual actions in parallel as there are workers. The workforce resource has a priority queue; the rule for requests in the queue with equal priority is FIFO.

Note that we model individual machines because each operation requires a specific machine. For the workers, due to the homogenous workforce qualification, a single resource with the capacity of the number of workers $|W|$ that is encoded in the roster is sufficient. Workers are interchangeable and they can switch freely between manual actions.

The resources are requested and released by the following processes:

Operation Processes: For each operation of the extended JSP instance, there is an *operation process* as follows:

- For each job j, the process for the first operation of j starts at α_j^{plan}, i.e., at the planned release (start) of the job. Thus, operation processes may run in parallel.
- An operation process, once started, will immediately request all machine resources of all modes of its operation – the above mentioned virtually queuing up – and then wait until at least one request is granted. This will happen as soon as the request is first in the machine resource queue (FIFO) when the machine resource is released by another operation process. Once at least one of the requested machine resources has been granted, the algorithm – simulating the decentralized control – selects the corresponding mode to the granted resource – namely the one with shortest processing time (SPT). All other requests of the alternative modes are canceled before the next time step of the DES is taken.
- After a mode has been selected, the operation process counts as many discrete time steps as the duration of the mode specifies – this simulates the running of the operation.
- If the selected operation mode contains a manual activity, the operation process will request a worker (one capacity unit from the workforce resource) as soon as the simulation event manager reaches the starting point of that manual activity. In case there is no worker available, the process waits and suspends the operation including the machine until a worker will become available. Once a worker is granted, the process continues counting time steps until the manual activity is over. It then releases the worker and counts the remaining steps for the machine operation of the selected mode (if any).
- Finally, the process releases the machine resource. If there are remaining operations within the job j, it starts the operation process for the next operation of job j. It terminates when there are no operations left.

The operation processes thus fix the start and end times of operations and manual activities.

Roster Process: The workforce resource has the capacity $|W|$. Our algorithm needs to adapt this capacity during simulation such that it represents the number workers actually deployed, not just the constant overall staff size |W|. Using the information about deployment intervals encoded in the chromosome, our algorithm computes for each point in time the number of workers

that should be deployed. During simulation, a single *roster process* requests and releases capacity from the workforce resource. It does this exactly at the times when the number of workers in the roster changes, such that these "ghost requests" allocate exactly the number of workers that are *not* deployed and should thus not be available to become allocated by operation processes.

- E.g., if the roster specifies that at the times t_1, t_2, t_3, there are $3, 4, 2$, respectively, of the $|W| = 5$ workers deployed, the roster process will request 2 workers at t_1, release 1 worker at t_2 and request 2 more workers at t_3.
- The roster process works with high priority against the workforce resource, so that it always supersedes requests made by operation processes.
- To deal with workforce rosters that deploy too few workers and cannot finish all manual operations, a further rule is required: When the very last deployment interval in the roster ends, the roster process does not request all workers, thereby leaving none deployed, but instead releases all workers forever so that they are all available for operation processes. Thus, the simulation is guaranteed to terminate (all manual activities will be completed). This situation will be accounted for by a penalty in the fitness function, reducing the fitness of workforce rosters that are too short to finish all jobs.

Note that once an operation process has been granted a worker (one capacity unit from the workforce resource), even the roster process cannot take it away. This corresponds to the feature that workers will finish their tasks (if there is no other worker to hand over the task) even though they might work overtime, which accounts for the differences between D_w^{plan} and D_w^{eff}.

4.4 Tournament Selection

As selection operator, we use tournament selection. By applying this selection mechanism, we can control the selection pressure [44].

Our GA selects the best individual out of a group (tournament) of, e.g., five randomly chosen individuals of the population. We repeat the selection process as many times as there are individuals in the population, thus keeping the population size constant across generations. The selected individuals are further processed by crossover and mutation to form the next generation. The size of the group – the tournament size – is configurable in our framework; for our experiments, we have had good results with size 5, see Sect. 5.

4.5 Genetic Operators: Crossover and Mutation

Our GA uses crossover (mating) as well as mutation operators that can handle the variable length of the chromosomes and the several worker sections. Due to these specialties, the mating operators must be designed accordingly which we describe in detail in Subsect. 4.5.1. However, we also use mutation operators that specifically target the chromosome length (number of decision variables). We define them in Subsect 4.5.2.

Directly after each genetic operation a repair algorithm (see Sect. 4.6) is applied, even before the target function evaluation. This repair algorithm changes an individual (i.e., roster candidate) such that it complies with the required constraints of working time laws.

4.5.1 Crossover for Recombination of Individuals

Firstly, a general probability is set that a mating process will take place. If there is a mating process, there will be two options of crossover operators. The algorithm executes exactly one of these two operators by applying their assigned probabilities, which add up to 1.

The first option is an adapted **messy crossover**:

A messy crossover [45] is typically a one-point crossover of the type "cut and splice" that does not require parent individuals of the same length. For each parent individual, a cut is made through the chromosome independently of the other parent. For the first child, the first half of the first parent is concatenated with the second half of the second parent. The second child is created the other way around. Correspondingly, the child individuals can vary in their lengths compared to each other and to the parents.

For our implementation, we modified the standard messy crossover scheme with respect to the individual chromosome sections of the workers. The individual worker sections are mated with the corresponding sections in the other parent. Accordingly, the messy crossover does not cross the chromosome as a whole but individually recombines the new solutions pairwise of chromosome sections of the respective worker.

The second option is an adapted **simulated binary crossover**:

In contrast to the messy crossover, a simulated binary crossover is not a "cut and splice"-operator, but merges the values of the parent chromosomes (for further information see, e.g., [46]). Using these different strategies gives us more potentially genetic development. However, for the simulated binary crossover, it is important that the two parent chromosomes be of equal length, i.e., have the same number of deployment intervals in corresponding worker sections, which is not the condition with our GA. Therefore, and with regard to the worker section within the chromosomes, we also modify this operator: To handle the chromosome lengths, we modify the parent chromosomes so their corresponding worker sections have the same length – at least temporarily. We apply the following procedure section by section for each worker: Firstly, the algorithm determines the difference in length for each worker section. Secondly, it removes the required number of randomly selected deployment intervals from the section with the longer length. As a result, the pairs of chromosome sections of each worker are of the same length. In the next step,

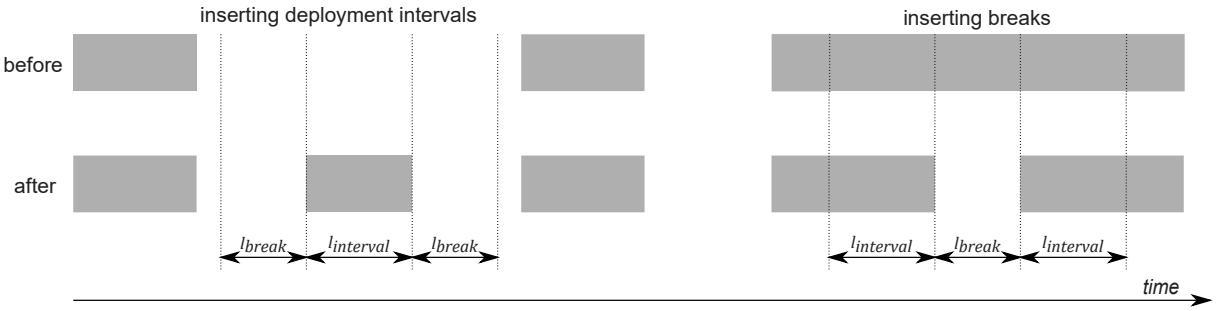

Fig. 4: Schematic illustration of the mutation operators "inserting deployment intervals" and "inserting breaks"

standard simulated binary crossover is performed for the chromosome sections. Finally, within the created children, we re-insert the previously removed deployment intervals by random decision (50% chance each). Deployment intervals not picked by this random choice will be discarded. Note that the place of re-insertion of the intervals is of no consequence, as our repair algorithm will sort each worker section anyway according to the starting point of the deployment intervals.

4.5.2 Mutation

Analogously to the crossover operators, the application of mutation operators can also be controlled with a parameter for the probability of performing a mutation at all. There are three different operators, but these work independently of each other.

The first mutation operator is a **Gaussian mutation** operator [41] that changes the values according to a normal distribution. For a small standard deviation, it focuses on an exploitation character and with a large standard deviation it focuses on an exploration character. We will focus on the exploitation. In our case, this operator receives four parameters: A probability to be applied for the selected individual for mutation, an independent probability for each chromosome value to be mutated, the mean and the standard deviation. This operator can be used as defined in the literature or in the framework DEAP and needs only a slight adaption to our chromosome formulation, i.e., it changes single values that can be a start time and/or a duration of a deployment interval. Since we are working with a discrete-event simulation, the generated values are rounded to integers.

The second mutation operator "insert deployment intervals" and the third mutation operator "insert breaks" target the length of the chromosomes, i.e., the number of decision variables and are not part of the existing framework DEAP. During the development of the methodology and especially in connection with the repair algorithm (see Sect. 4.6), we observed that the chromosome length tends to become shorter and shorter over the generations. We decided to counteract this by the possibility of mutation – more precisely, by inserting additional deployment intervals as well as splitting existing deployment intervals by inserting a break. We also expect that these mutation operators will make it easier for the GA to overcome the highly fragmented search space boundaries between search space segments (exploration).

The second and third mutation operator work the same way but complementary. Each of them has a parameter of probability to be generally applied at an individual and a parameter of the probability of inserting a deployment interval or a break, respectively, at a specific point. Both operators work within the boundaries of a worker section within the chromosomes.

The mutation operator "**inserting deployment intervals**" scans each time period in the chromosomes where no working hours are scheduled, i.e., each gap between deployment intervals. If the gap is long enough, then depending on the insertion probability, a new deployment interval may be inserted. A gap is long enough if it is at least of the duration of two minimal breaks plus a minimal deployment interval (see Fig. 4). The operator does not take into account whether the gap represents a (short) break or a (longer) resting period. The inserted deployment interval is of the minimal allowed length and is inserted at the middle of the gap. Thus, the new deployment interval keeps a distance to existing deployment intervals of at least the minimal break duration.

The mutation operator "**inserting breaks**" works the other way round. It scans the deployment intervals within the chromosome. If an interval is long enough, then depending on the insertion probability, a break may be inserted into the interval. An interval is long enough if it is at least of the duration of two minimal deployment intervals and a minimal break (see Fig. 4). The inserted break is of the minimal allowed break length and is inserted in the middle of the original deployment interval. Thus, it splits the original interval into two new deployment intervals of at least minimum length.

4.6 Repair Mechanism

Due to the many restrictions from working time legislation, our problem of finding an optimal roster is a highly constrained problem with a very fragmented search space of the decision variables. In fragmented solution spaces, the optimization strategy faces the difficulty of generating new solution proposals, which overcome the segment boundaries and lie in other solution space segments than the known solutions (e.g. the evaluated parent generations of the GA) (for further information see, e.g., [47]). In addition, in our case the constraints of a decision variable are not fixed but depend on the values of the other decision variables, at least within a worker section. Furthermore, there is no fixed number of decision variables at all.

Accordingly, in the procedure of stochastic evolution, the probability is high that crossover and mutation operators violate constraints and thus the GA creates invalid child individuals. A key aspect in the development of our solution algorithm is therefore the question of how to deal with the constraints and/or the constraint violations. In the literature, one can find different existing constraint-handling techniques for GA. These are, among others, especially penalty functions and repair mechanisms [48]. We do not want to have unacceptable solutions for the roster creation at the end of the optimization process, but want to be sure that the roster is definitively valid for all relevant labor time regulations ("hard constraints"). When using penalty functions, death penalty is a way to be sure of evolving generations with valid solutions. In this strategy, all invalid solutions receive such a high penalty that they are completely uninteresting for the further optimization process [48]. Given the large number of constraints, we assume that stochastic evolution does not produce a large number of valid individuals and the loss of too many individuals strongly affects the search mechanism of the GA within the solution space. Instead of rejecting invalid solutions by high penalty costs, we are going to repair them. This way we can be sure to always get valid solutions (=rosters) in the sense of the working time regulations. With our repair algorithm, we want to change the individuals as little as possible and we are only focusing on the constraints of working time restrictions. Due to the high number of solution proposals and function evaluations in connection with the high number of constraint violations we designed the repair algorithm to be fast and easy applicable. The developed repair algorithm consists of five main repair operators (MRO) with a fix application sequence and two auxiliary repair operators (ARO) (see Fig. 2). The mechanism ensures that all constraints are satisfied step by step. By applying the main repair operators one by one, there is one by one more constraint met. The main operators are designed in such a way that the established condition of constraint satisfaction of the previous operator is not violated. Each main repair operator needs to be applied only once per individual. The auxiliary repair operators are applied at the beginning and after each main operator to maintain a correct chromosome formulation.

Each constraint is enforced individually for each worker section in the chromosome. All repair operators are applied within a worker section and independently from other worker sections.

Auxiliary Repair Operator 1: Round & Sort

This operator is the first step of maintaining a correct chromosome formulation and is only applied at the beginning of the repair algorithm. It guarantees the pre-condition for the remaining repair operators that all deployment intervals contain only integers and are sorted. For example, if one of the genetic operators produces a non-integer (in violation to our problem formulation), this operator corrects it to an integer. Additionally, the operator sorts the starting points of the deployment intervals within a worker section in ascending order. For example, the crossover operators can cause non-sorted deployment intervals. In particular, a random creation of the initial individuals needs the sorting process.

Example: The two deployment intervals ([2, 6.9], [0.3, 5]) are replaced by ([0, 5], [2, 7]). This is not yet a correct chromosome representation as defined before. The description of the next auxiliary repair operator shows the missing modification.

Auxiliary Repair Operator 2: Merge

This second auxiliary operator keeps the representation of working time hours per worker unique and maintains a correct chromosome formulation. It is applied after the first auxiliary repair operator and after each main repair operator. It adjusts overlapping and "touching" deployment intervals within one worker section. With overlapping intervals, the end time of the first intervals is later than beginning of the next interval. With touching intervals, the end time of the first interval is the start point of the second interval. In both cases, the operator merges them into a single interval with start time of the first interval and end time of the second interval. A correct problem formulation does not contain any overlapping or "touching" intervals. The genetic operators and random generation of individuals, as well as the five main repair operators themselves can cause such situations of overlapping or "touching" intervals.

Example: The two deployment intervals ([0, 5], [2, 7]) are replaced by the single deployment interval ([0, 7]).

Main Repair Operator 1: Minimal Interval Duration $l_{interval}$

This operator fixes violations of the constraint of minimal duration of a deployment interval (see Fig. 5). It does not consider the other constraints. Deployment intervals that are below a configurable threshold $l_{interval}$ will be extended (see Fig. 5 case 1). The extension take place at the end of the interval, so the

end will be shifted towards the future. In this way, an interval can be merged with the following interval (see Fig. 5 case 2). This is the only main repair operator that extends the working hours of the workers.

In our application, we set the lower bound $l_{interval} = 1h$.

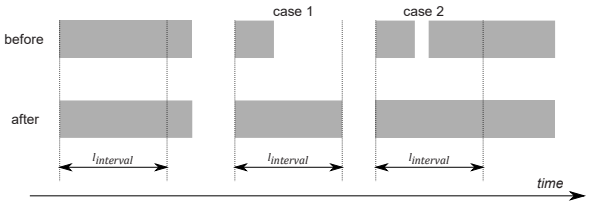

Fig. 5: Schematic illustration of MRO 1 (minimal interval duration $l_{interval}$)

Main Repair Operator 2: Maximal Interval Duration $u_{interval}$

This operator manipulates the duration of the deployment intervals in view of a configurable upper limit of duration, after which the worker has to take a mandatory break. After the maximum time for a deployment interval $u_{interval}$, a break is inserted that lasts for the configurable minimum duration of breaks l_{break}. This means that, if necessary, deployment intervals are split up (see Fig. 6 case 1). In order to avoid the destruction of the previously established minimum working times of the intervals by main repair operator 1, resulting intervals with a length below the minimum length are completely deleted (see Fig. 6 case 2).

In our application, we set the upper bound $u_{interval} = 6h$ to comply with the German Working Time Act.

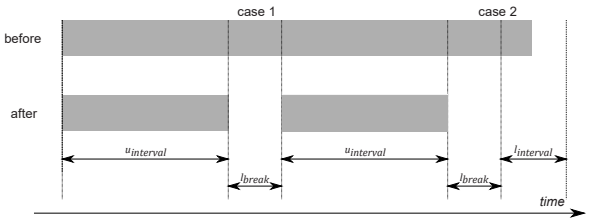

Fig. 6: Schematic illustration of MRO 2 (maximal interval duration $u_{interval}$)

Main Repair Operator 3: Minimal Break Duration l_{break}

Once the worker has worked a certain amount of time (e.g., see MRO 2), a break should be taken. At this point, we make a simplification for the algorithm. All designated breaks shall be of the specified minimum duration. That means we do not check whether we can piece together the necessary break time from several breaks (as would be OK with working time regulations), but make sure that each break is already long enough on its own for this shift.

Concretely, the operator works like this: If a break is too short, the start of the next deployment interval is moved to a later point in time (see Fig. 7 case 1). Then, if necessary, we have to remember our condition that we will not harm the already established constraints. If the following deployment interval gets too short, the operator will delete it completely (see Fig. 7 case 2).

In our application, we set the lower bound $l_{break} = 1h$.

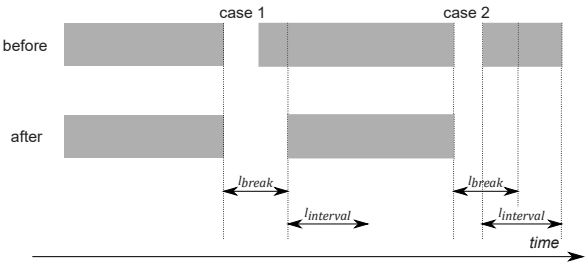

Fig. 7: Schematic illustration of MRO 3 (minimal break duration l_{break})

Main Repair Operator 4: Maximal Number of Working Hours per Shift u_{shift}, Maximal Span of Shift $u_{shiftspan}$ and Minimal Rest Period Duration l_{rest}

This operator combines the observance of three constraints that have strong interdependencies between each other. These are the maximal amount of work within a shift, the length of a shift and the minimal rest period that divides two shifts (see definitions in Sect. 3). Therefore, it is the most complex repair operator which is modeled in a recursive way and uses two explicitly modeled global states in its algorithm.

Repair operator 4 goes through the list of intervals of an individual and deletes or shortens intervals according to the following criteria. We explain the process on the base of our configured values (see Table 2):

1. The first deployment interval of the schedule of a worker starts a shift and the operator enters the state **"in-shift"**. This means that the next at most 13 h (max. shift length $u_{shiftspan}$) of the time line are interpreted as a shift. The shift might end earlier if the operator encounters a period without deployment intervals of at least 11h (minimal rest period l_{rest}).
2. **In the state "in-shift"** the algorithm keeps track of the amount of work (the sum of the duration of deployment intervals) within the shift and handles the upcoming deployment intervals one by one. It distinguishes three cases:
   2.1. **The current deployment interval is completely within the considered shift, i.e. within the range of $u_{shiftspan}$:** If the deployment interval (potentially together with previous intervals of the same shift) exceeds the maximum amount of work u_{shift} (see Fig.

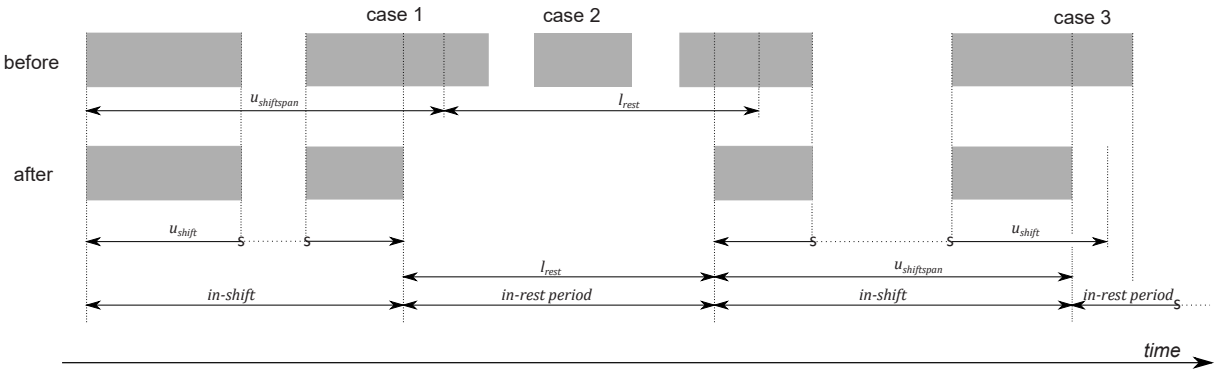

Fig. 8: Schematic illustration of MRO 4 (restrictions regarding shift workload u_{shift}, shift span $u_{shiftspan}$ and rest period duration l_{rest})

8 case 1), it is truncated accordingly and the operator switches its state to **"in-rest-period"** (see step 3). Otherwise, the algorithm stays in the state **"in-shift"** and proceeds with the next deployment interval.

2.2. **The current deployment interval is completely outside the considered shift, i.e. outside of the range of $u_{shiftspan}$:** This means that the end of the previous deployment interval has ended the shift and the next mandatory 11h (l_{rest}) rest period has already started then (see Fig. 8 case 3). The operator switches its state to **"in-rest-period"** (see step 3) and handles the unaltered current interval recursively under this new state. (The interval might lie within the rest period and has to be discarded completely or it might be so far in the future that it will constitute a new shift – the operator handles these cases recursively.)

2.3. **The current deployment interval overlaps with the end of the considered shift, i.e. is partly inside and outside of the range of $u_{shiftspan}$:** In this case (see Fig. 8 case 3) the algorithm splits the deployment intervals in two at the edge of $u_{shiftspan}$ ("touching" deployment intervals are allowed during the application of this operator). Thus, the operator creates a situation in which step 2.1 and 2.2 can be applied. It handles both new deployment intervals recursively under the current state (**"in-shift"**) as described in step 2.1. and 2.2.

3. In the state **"in-rest-period"**, the operator will clear for an 11h (l_{rest}) rest period starting with the end of the deployment intervals that ended the previous shift. It again distinguishes three cases:

3.1. **The current deployment interval is completely within the range of l_{rest}:** This interval is simply discarded (see Fig. 8 case 2).

3.2. **The current deployment interval is completely outside the range of l_{rest}:** This interval starts a new shift. The operator changes its state to **"in-shift"** and proceeds recursively with the current deployment interval (thereby processing the interval again but under a different state).

3.3. **The current deployment interval overlaps the end of the rest period of duration l_{rest}:** Here, the operator splits the interval at the edge of l_{rest}. Thus, the operator creates (like in step 2.3) a situation that allows for recursive resolution. The operator now handles both new deployment intervals recursively under the current state (**"in-rest-period"**) on the basis of steps 3.1 and 3.2.

Main Repair Operator 5: Maximal Working Time per Day u_{day}

This operator ensures that the maximum allowed daily working time u_{day} is not exceeded. After the configured threshold of working hours in a day, all further working hours on that day are cancelled (see Fig. 9 case 1). In other words, after summing up the working time with starting from the beginning of the day, a cut is made after the reached amount of the configured hours (in our case, e.g., 8 h). All later deployment intervals of that day are deleted. If the threshold is reached within a deployment interval, the interval will be shortened. If the threshold is reached within an interval whose end lies in the next day, the interval will be shorted until the new day starts but at least by a minimal break length. Generally, if a split or shortened deployment interval gets too short, the operator will delete this interval of too short duration (see Fig. 9 case 2).

In our application, we set the upper bound $u_{day} = 8h$. According to this concept, the realization of a weekly or longer limit is also conceivable.

Summary of the Main Concept of the Repair Operators

The auxiliary operators keep the correct format of the chromosomes. The main operators control the working time restrictions and modify the working hours, if necessary. They have a fixed application sequence.

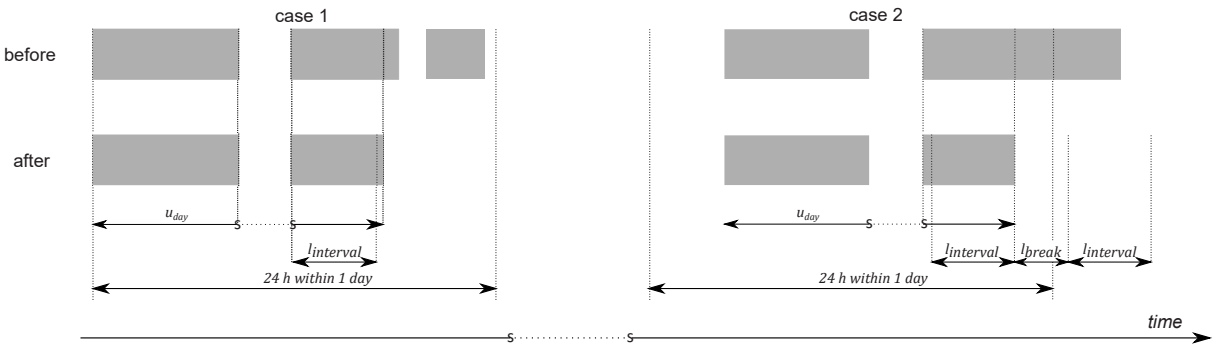

Fig. 9: Schematic illustration of MRO 5 (maximal working time per day u_{day})

Applying one by one, we can be sure that there will be one more constraint met. Only the first repair operator will increase the working hours by extending deployment intervals. All other operators have a shortening effect on the working times by shrinking deployment intervals. Thus, operators 2-5 will keep the constraints automatically intact except for constraint of operator 1. In order to keep constraint of operator 1, intervals of too short duration will be discarded at the end of the application of every repair operator.

5 RESULTS

In this section, we describe the used problem instances, followed by the results of the experiments.

5.1 Problem Instances

To the best of our knowledge, there are no benchmark instances that fit our problem statement. We do not want to use completely new and unknown instances, but we will extend existing benchmarks. As basis, we choose well-known flexible job shop scheduling instances, which we extend for time windows and manual tasks. Release date α_j^{plan} and due date ω_j^{plan} (the time window) are generated by two parameters, a concurrency factor $k_{concurreny}$ and a due date factor $k_{DueDate}$, respectively. Both parameters calculate the respective time based on the shortest possible execution time CP_j of every job j, i.e., the shortest possible path across the various operations:

$$\alpha_{j+1}^{plan} = \alpha_j^{plan} + (CP_j * k_{concurreny}) \qquad \forall j \in J, j > 1, \quad (8)$$

where the release date of the first job $\alpha_1 = 0$. For example, the concurrency factor $k_{concurreny} = 0$ means that all jobs start at the same time $t = 0$. $k_{concurreny} = 1$ means that if every job were done in its shortest possible time directly at its release date, there would be no parallel jobs.

$$\omega_j^{plan} = \alpha_j^{plan} + (CP_j * k_{DueDate}) \qquad \forall j \in J. \quad (9)$$

For example, the due date factor $k_{DueDate} = 1$ means that the due date is so tight that the job can be finished in time only when the shortest operation mode is selected every time and there are no waiting processes.

The extension with manual tasks is a random-based process. It is controlled by the parameter p_{manual} that is the probability that a machine operation is assigned a manual task. The exact start of the manual task within the machine operation and the duration is randomly generated, but is limited by the length of the machine operation. For example, the probability $p_{manual} = 0.5$ means that half of the machine operations receive a manual task. However, it does not mean that the workload of all manual task together is half the workload of all machine operations (due to the shorter duration of the manual tasks).

We choose the number of workers in a way that with a common 2- and 3-shift system the instances can be solved neither with much delay nor with a large staff surplus.

As a final point of understanding the instances, the temporal interpretation must be clarified: Since we are creating rosters for workforce management, we will interpret the dimensionless time units of the instances to contain the workload for approximately one week (ca. 5-7 days).

For our experiments, we use the instances Behnke & Geiger 60 [49], Brandimarte Mk15 [50], Dauzere 15a [51] and Fattahi 20 [52] with the modification parameters shown in Table 3.

Table 3: Modification parameters of the used benchmark instances

Behnke & Geiger 60	$k_{concurreny}$	0.2	
	$k_{DueDate}$	2	
	p_{manual}	0.5	
	$\|W\|$	4	
	Time unit interpretation	5	[min]
Brandimarte Mk 15	$k_{concurreny}$	0.1	
	$k_{DueDate}$	2.5	
	p_{manual}	0.6	
	$\|W\|$	6	
	Time unit interpretation	10	[min]
Dauzere 15a	$k_{concurreny}$	0	
	$k_{DueDate}$	3	
	p_{manual}	0.4	
	$\|W\|$	3	
	Time unit interpretation	3	[min]
Fattahi 20	$k_{concurreny}$	0.05	
	$k_{DueDate}$	3	
	p_{manual}	0.8	
	$\|W\|$	5	
	Time unit interpretation	4	[min]

Detailed Example: Brandimarte Mk15
Fig. 10 shows the 30 jobs of the Brandimarte MK15 in a Gantt-style diagram. The offset of the start times of the jobs results from the configured job concurrency and expresses the initial production plan of not beginning all jobs at once. The dark grey bars visualize the shortest possible execution time of the respective job, i.e., the case when for each operation the shortest mode

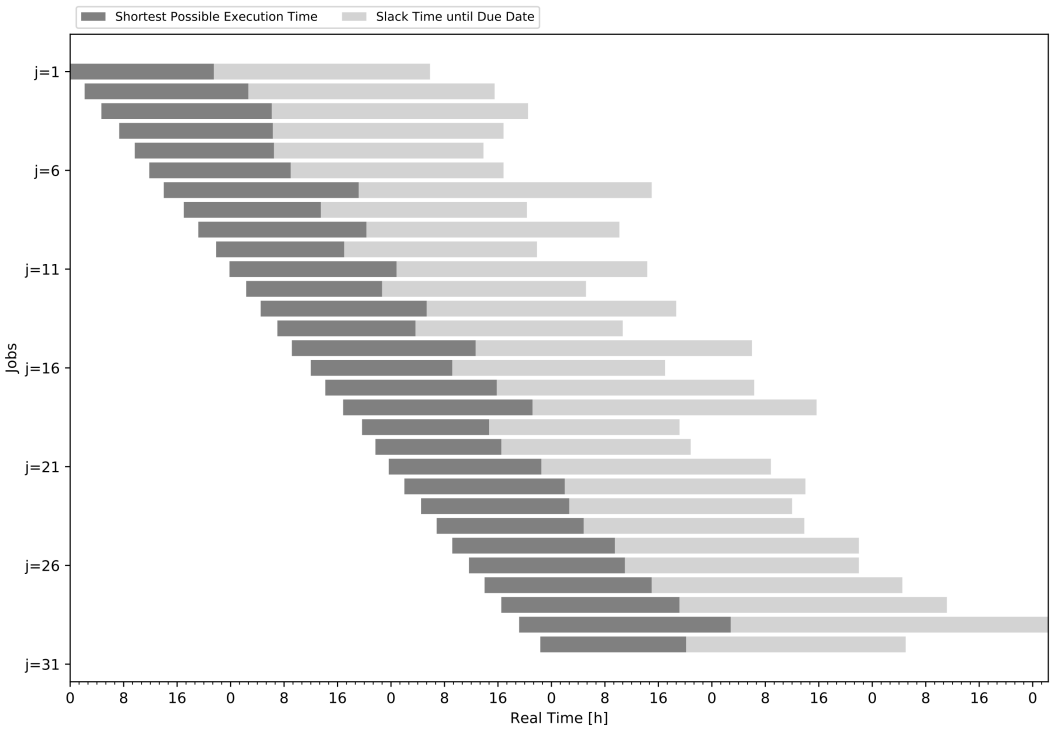

Fig. 10: Release date α_j^{plan} due date ω_j^{plan} and shortest possible execution time CP_j of job j for test instance Brandimarte Mk15

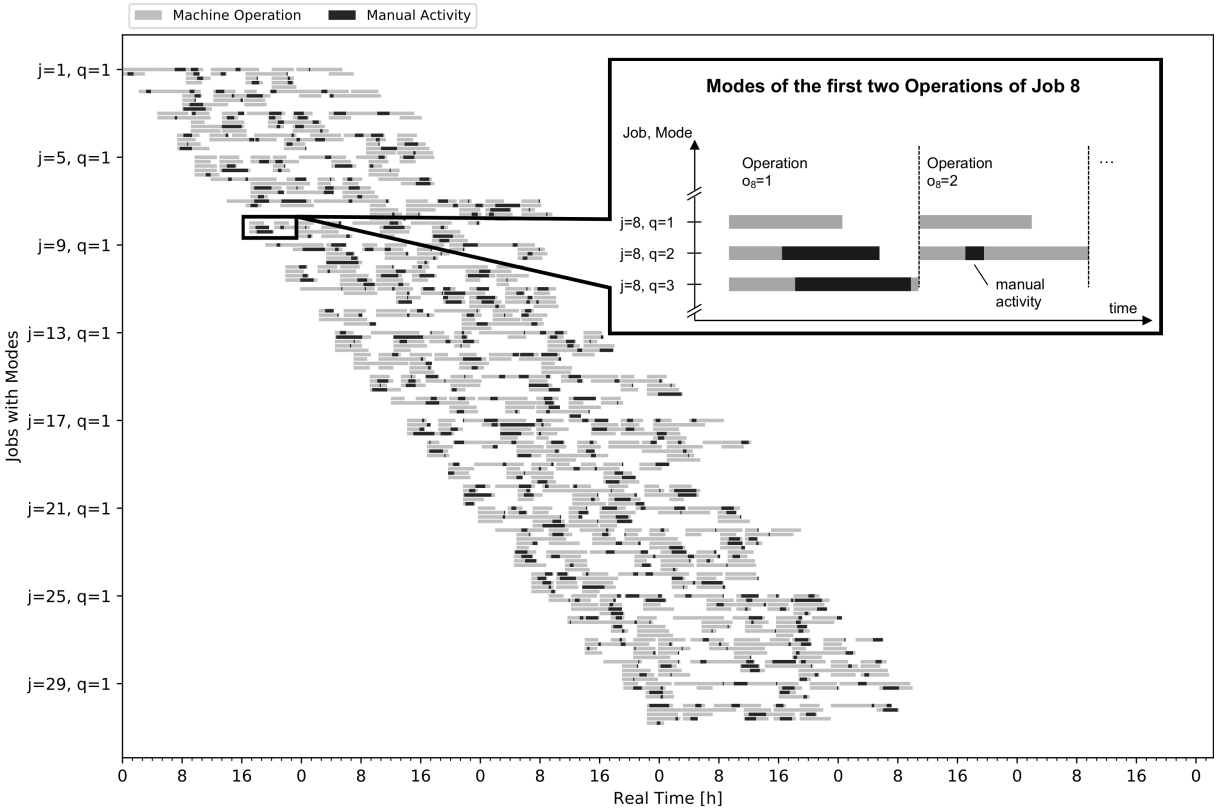

Fig. 11: Illustration of the share of manual activities (in black) as well as alternative modes of the jobs for test instance Brandimarte Mk15

would be chosen. The light grey bars visualize the designated slack time until the due dates of the jobs. If a job finishes before its due date, there will not be any delay costs.

Fig. 11 shows the same problem instance in more detail and without the due date information. Each of the 30 vertically arranged jobs consists of averagely three operation modes which are represented by the horizontal lines. The horizontal segmentation of a job in operations is visualized by the gaps in the horizontal lines – the longest mode of an operation determines the length of the line and the gaps appear for the modes with shorter duration. The dark line segments visualize manual activities of which each operation mode can have at most one.

5.2 Evaluation

Since there are no existing benchmark results available from other publications, we will compare the performance of the developed algorithm to randomly generated solutions as well as to the two- and three-shift systems widely used in practice. All experiment runs are based on the same number of available workers in each respective instance. In this way, all compared rostering methods have the same upper bound of working hours capacity but differ in the arrangement of working hours (i.e., the roster). Keep in mind, that the number of workers is set in a way that the 2- and 3-shift system can solve the instances neither with much delay nor with a large staff surplus (see Sect. 5.1).

The randomly generated solutions are not completely blindly generated. We generate them in the same way we generated the start population (see Sect. 4.5) and guarantee their feasibility by applying the repair rules. The number of the randomly generated solutions is equal to the number of individuals (candidate rosters) of the proposed GA framework, so there will be the same number of objective function evaluations in both test runs.

In the two- and three-shift systems, the workers are equally distributed over the shifts. If the number of workers is not divisible by the number of shifts, they are assigned to the morning shift and to the evening shift, if necessary. The shift times are as follows: 6:00 a.m. to 2:00 p.m. (morning shift), 2:00 p.m. to 10:00 p.m. (evening shift) and 10:00 p.m. to 6:00 a.m. (night shift). Accordingly, there is only one objective function evaluation for each test setup, as the roster is already fixed.

The desired working times are the same for all experiments and are generated per worker for each day as an 9 h-interval (1 hour lunchbreak included), which is normally distributed with mean at 8:00 a.m. and with a standard deviation of 2 h.

Table 4: Applied parameter set for the evaluation process

Parameter set of GA		Parameter set of cost function	
Size of population	100	Labor cost [€/h]	25
		Night hours [clock hours]	[23, 6]
Number of generations	200	Night work surcharge [%]	25
Stop after generation without min. improvement	100		
With minimal improvement [€]	1	Surcharge non-preferred working hours [%]	25
Tournament size	5	Job delay cost [€/h]	100
Mating Probability	0.8	Penalties	
Probability of messy crossover	0.5	Unplanned work penalty [%]	200
Probability of simulated binary crossover	0.5	After-schedule work penalty [%]	300
Eta of simulated binary crossover	2		
Mutation Probability	0.1		
Probability of Gaussian mutation	0.05		
Sigma of Gaussian mutation [minutes]	120		
Probability of mutation “inserting break”	0.05		
Probability of mutation “inserting interval”	0.05		

Table 4 shows the applied parameter set for the evaluation process. In this experiment we used only one configuration. We have determined this parameter combination by a grid search in preliminary tests. This combination has shown good results in most experiments, although for single instances other parameter compositions can give better results.

Fig. 12 shows the results of the comparison for the four selected instances. The randomly generated start population of the proposed algorithm is not yet necessarily better than the two- and three-shift operation and it takes some generations to achieve a better cost value. However, our developed algorithm achieves the best results in our test instances, provided the number of generations is high enough. The random solution generation (analogous to the start population generation) beats the rigid shift systems after a certain number of trials in three of the four cases.

Table 5 shows the objective function evaluation broken down into the three main cost aspects and the penalty due to workers that have worked longer than rostered.

The delay costs tend to be highest in the 2-shift system. A detailed analysis reveals that in the two-shift system, there is a backlog of manual task (and thus of jobs) at night. At the beginning of the morning shift the workers process the waiting task of the night and thus the worker capacity utilization is high. Then the utilization decreases until the end of the second shift. In the three-shift-system, there is a consistent supply of workers so that the backlog does not accumulate as in the two-shift system. However, on the one hand, it is not possible to respond to peaks of capacity demand due to rigid capacity supply. On the other hand, there are long waiting times without tasks for the workers that create periods of low worker capacity utilization. Accordingly, the rigid shift systems have clear disadvantages, which do not exist for our algorithm that supports flexible working hours and is adaptable to hourly volatile capacity demands. A detailed evaluation of the values shows that our proposed algorithm generally schedules less attendance time and the deployment intervals are better adapted to hourly volatile demands.

Moreover, the rigid shift systems have both the most expensive costs in the objective function part of desired working hours of workers. Neither in the 2-shift nor in the 3-shift system, the rostering process is based on the inclusion of desired working times. The overlap between desired working times and actual working times is only by chance. The same applies to the randomly generated solutions. However, these perform significantly better in comparison; possibly, because they can also guess outside fixed shift system time grids. Our algorithm is on a similar level of costs for rostered non-desired working hours as the randomly generated solutions.

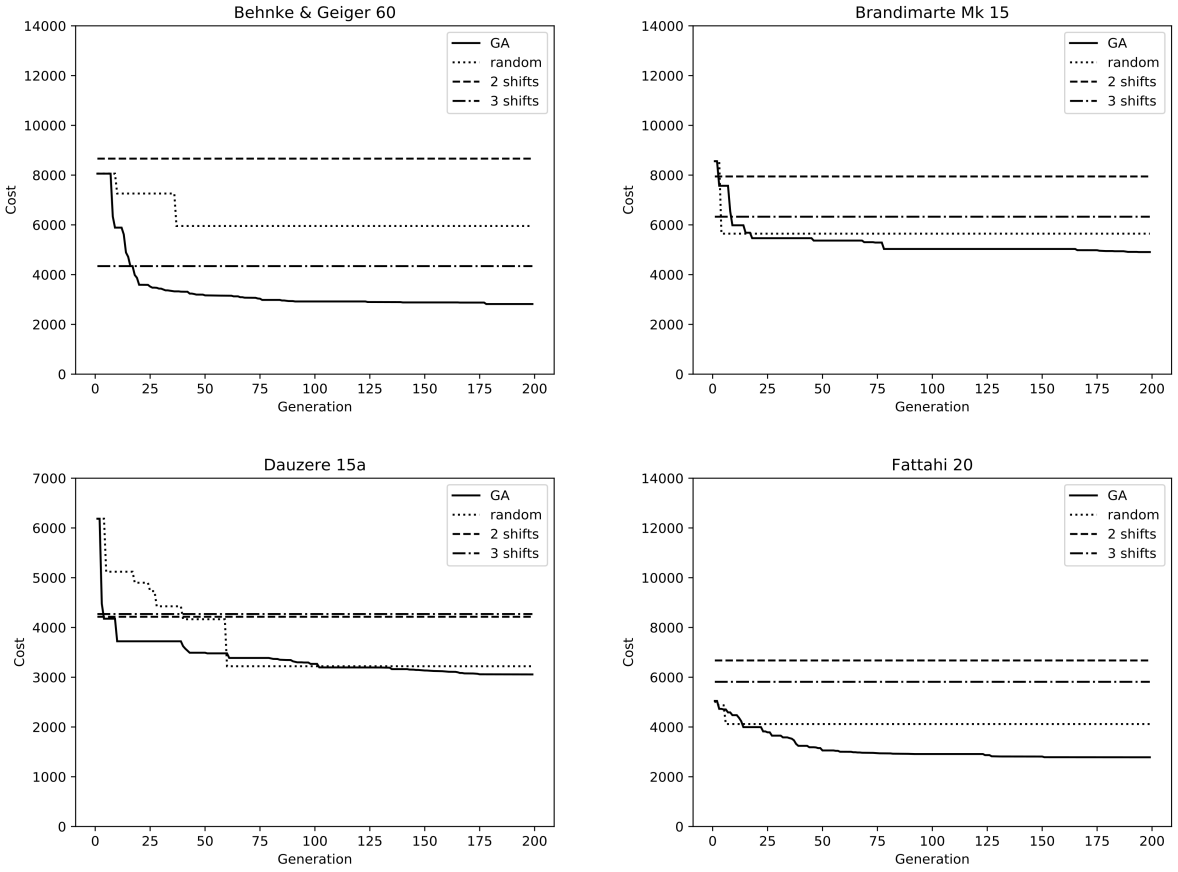

Fig. 12: Cost evaluation (fitness function including objective function and penalties) of developed algorithm in comparison to two- and three-shift system and randomly generated solutions for the test instances

Table 5: Cost values of the best solution found

		$c_{fitness}$ [€] (rounded)	$c_{objective}$ [€] (rounded)	$c_{attendance}$ [€] (rounded)	$c_{undesired}$ [€] (rounded)	c_{delay} [€] (rounded)	$c_{penalty}$ [€] (rounded)
2-shift system	Behnke & Geiger 60	8664	8651	3590	444	4617	13
	Brandimarte Mk 15	7945	7541	6202	656	683	404
	Dauzere 15a	4232	4127	3688	439	0	105
	Fattahi 20	6675	5913	5436	477	0	762
3-shift system	Behnke & Geiger 60	4341	4333	3629	446	258	8
	Brandimarte Mk 15	6328	6328	5533	795	0	0
	Dauzere 15a	4257	4257	3663	594	0	0
	Fattahi 20	5816	5779	5116	663	0	37
By random	Behnke & Geiger 60	5954	5773	2799	357	2617	181
	Brandimarte Mk 15	5646	4986	4236	433	317	660
	Dauzere 15a	3221	2813	2483	330	0	408
	Fattahi 20	4119	3365	2992	373	0	754
Proposed algorithm	Behnke & Geiger 60	2818	2801	2419	374	8	17
	Brandimarte Mk 15	4907	4653	4093	560	0	254
	Dauzere 15a	3057	2694	2410	284	0	363
	Fattahi 20	2784	2526	2198	328	0	258

We have set the initial time horizon for every experiment run as long as necessary so that in the best evaluation runs there are no costs for too short rosters $c^{penalty}_{afterschedule}$. Accordingly, the penalty costs $c_{penalty}$ in Table 5 are the same as the penalty costs for unplanned working hours $c^{penalty}_{unplanned}$.

A disadvantage of our algorithm in comparison to rigid shift systems is a higher expensiveness of workforce roster creation. The algorithm takes many target function evaluations (e.g., for a population size of 100 individuals evolving over 200 generations: 20,000 cost functions evaluations[2]). However, each cost function evaluation, which includes the simulation of the production system, takes only a fraction of a second for each of the four considered instances, so that on one CPU we can evaluate 20,000 individuals in a matter of minutes. There is of course potential for parallelization. Up to now, we have used parallelization only for grid search for GA parameter combinations. Accordingly, a scaling of the method to larger instances (more jobs / workers / machines) seems to be possible. However, to evaluate the method scientifically, correspondingly large benchmark instances will be needed.

[2] Usually, in our framework, an objective function evaluation corresponds to a simulation run. However, in the optimization process of the GA, identical individuals can arise, for which we re-use the already known cost values without another simulation run. Thus, typically, the number of simulations is somewhat smaller than the number of created individuals.

Detailed Example: Brandimarte Mk15

Fig. 13 displays information about the simulation run with the best individual, i.e., the best roster, found by the GA. The top chart shows the actual modes chosen for each job during the simulation and the time of their start and finish. The second chart visualizes the roster itself. It shows only the number of workers deployed at each time point. Note that the more detailed information which worker is deployed at which time can be read off the individual, see below. The third chart shows the actual time when workers were busy working. The

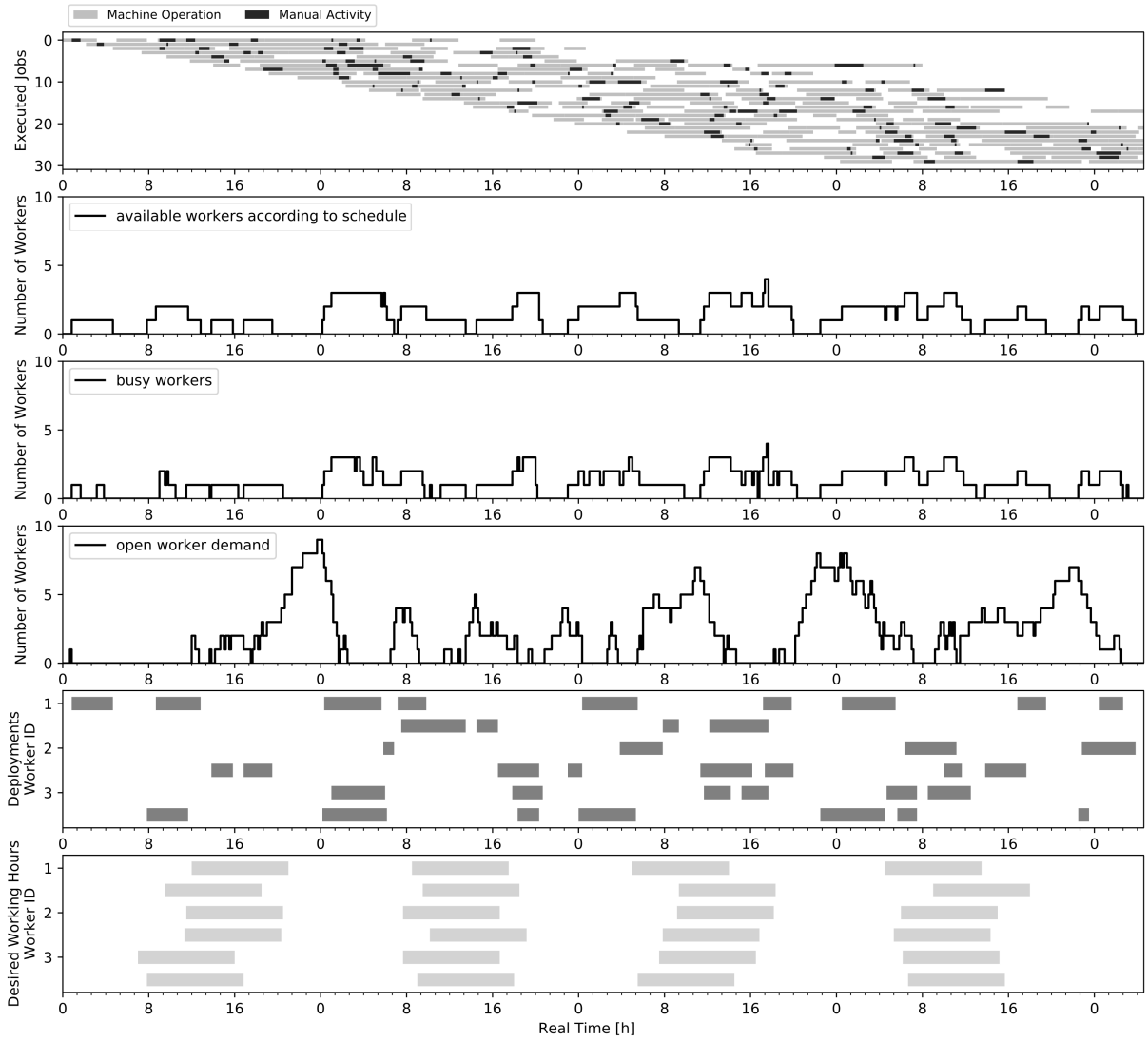

Fig. 13: Best solution proposal of our algorithm for test instance Brandimarte Mk15 (first four days)

fourth chart shows the unsatisfied or open demand for workers, i.e., the number of manual tasks that are suspended for lack of available workers. The fifth chart provides more detailed information about the rostered deployment intervals for each worker as Gantt-Chart. The bottom chart shows the desired working hours of each worker per day.

The solution generated by our algorithm (Fig. 13) shows that there may well be a backlog of manual task (and thus of jobs) during several hours (e.g., ca. day 1 at 6:00 p.m. until day 2 at 2:00 a.m. or day 3 at 8:00 p.m. until day 4 at 7:00 p.m.). In this example, three of four accumulated unsatisfied demand peaks are during the night. The scheduling of night work may be uninteresting for the algorithm for two reasons: The night work surcharge and penalty for undesired working hours (the desired working hours usually are during the day). It is cost efficient that the specified time window from job release to due date (and the resulting slack time) is used to improve the other two cost aspects. The algorithm seems to be able to use this effect.

The second (the number of workers deployed in the roster) and third (the number of busy workers in the simulation) chart of Fig. 13 have a high coverage rate that report a high capacity utilization of the worker. In simple words: If workers have been rostered, they are usually needed for processing manual tasks in the simulation.

6 CONCLUSIONS

The changed conditions related to Industry 4.0 as well as Industry 5.0 and Work 4.0 as well as the increased importance of work-life balance create a need for new methods for workforce rostering. For example, the integration of desired working hours of production employees is increasingly coming to the fore. Moreover, detailed baseline production schedules will become obsolete by the use of decentralized production control and can no longer be used as basis for conventional rostering methods.

With the changed conditions in mind, we proposed a new algorithm for workforce rostering in decentrally controlled production systems taking into account the desired working hours of the workers. The schedules generated are no longer based on rigid shift systems, but take advantage of flexible working hours. The core idea of the proposed solution is a simulation-based optimization method that uses a discrete-event simulation of the execution of jobs in a given production system and a specifically tailored genetic algorithm to generate cost efficient rosters. Workforce planning problems are highly constrained due to legal requirements. We counter this fact by using specially developed repair operators that modify infeasible solutions to feasible ones. Moreover, our genetic algorithm works with a variable chromosome length since the number of decision variables may vary depending on the solution. For the evaluation, we use well-known job shop instances which we have extended to the expected conditions since Industry 4.0. As our experiments show, the proposed algorithm achieves significantly better objective function values in comparison to ordinary two-shift and three-shift systems.

We have evaluated problems with up to 60 machines, 100 jobs with 5 operations each, and 10 workers. In all our experiments we observed a convergence towards a (at least locally) optimal fitness value after approx. 80-180 generations, see Fig. 12. We did not encounter any non-linear behavior in run-time, which is plausible, given our implementation of JSP simulation and the GA. These results indicate that our heuristic approach indeed renders our optimization problem tractable.

A key feature of our algorithm is that it supports the transition from rigid shift time grids towards flexible working hours. Accordingly, the supply of worker capacity (rostered attendance time) can be adjusted exactly to the capacity demands. Thus, a high working time efficiency is reached (not least against the background of sporadically occurring manual tasks in Industry 4.0). Moreover, the use of slack time with regard to delay costs helps to improve the other cost aspects (e.g., avoiding night work surcharge and employees working at undesired hours). Additionally, the developed algorithm meets the zeitgeist with regard to work-life balance as it takes into account preferred working hours of workers during the optimization process.

We continue working on our framework. The main new features currently under development are the support for heterogeneous skill levels of the workers as well as for stochastic process duration and robustness for process disturbances. These topics are also of high importance as can be seen from Sect. 1.1. In the future, we would also like to solve large-scale problems in order to be able to create adequate solutions for large industrial production systems. Another open topic is synchronization of existing rosters in case of last-minute changes (e.g., sick leave) that can occur frequently in workforce management.

Other useful extensions could be the assurance of minimum working hours for the workers or the detailed consideration of psychological and work science aspects. Legal aspects also need to be clarified and adapted to specific countries before using the methodology in companies.

ACKNOWLEDGEMENTS

We would especially like to thank the German Research Foundation / Deutsche Forschungsgemeinschaft (DFG), which is funding our project with the title “A simulation-based and flexi-time applying prediction model for scheduling personnel deployment times in the production planning process of cyber-physical systems” (project-id: 439188616).

REFERENCES

1. Schwemmer J, Schmidt T, Völker M (2020) A New Simulation-Based Approach to Schedule Personnel Deployment Times in Decentrally Controlled Production Systems. In: SIMUL 2020. Porto, Portugal, pp 19–23
2. Ittermann P, Niehaus J, Hirsch-Kreinsen H (2015) Arbeiten in der Industrie 4.0: Trendbestimmungen und arbeitspolitische Handlungsfelder. Study Hans-Böckler-Stift 308:90
3. Ganschar O, Gerlach S, Hämmerle M, et al (2013) Produktionsarbeit der Zukunft – Industrie 4.0: Studie. Fraunhofer-Verl, Stuttgart
4. Kühn M, Völker M, Schmidt T (2020) An Algorithm for Efficient Generation of Customized Priority Rules for Production Control in Project Manufacturing with Stochastic Job Processing Times. Algorithms 13:337. https://doi.org/10.3390/a13120337
5. European Commission. Directorate General for Research and Innovation. (2021) Industry 5.0: towards a sustainable, human centric and resilient European industry. Publications Office, LU
6. German Federal Ministry of Education and Research (2017) Industrie 4.0 | Innovationen für die Produktion von morgen. 11055 Berlin
7. German Federal Institute for Occupational Safety and Health (2019) Flexible Arbeitszeitmodelle – Überblick und Umsetzung. 44149 Dortmund
8. Bergmann F (2019) New work, new culture: work we want and a culture that strengthens us. Zero Books, Winchester, UK ; Washington, USA
9. Paul M, Knust S (2015) A classification scheme for integrated staff rostering and scheduling problems. RAIRO – Oper Res 49:393–412. https://doi.org/10.1051/ro/2014052
10. Artigues C, Gendreau M, Rousseau L-M, Vergnaud A (2009) Solving an integrated employee timetabling and job-shop scheduling problem via hybrid branch-and-bound. Comput Oper Res 36:2330–2340. https://doi.org/10.1016/j.cor.2008.08.013
11. Schwemmer J, Kühn M, Völker M, Schmidt T (2022) Scheduling Workforce in Decentrally Controlled Production Systems: A Literature Review. In: Freitag M, Kinra A, Kotzab H, Megow N (eds) Dynamics in Logistics. Springer International Publishing, Cham, pp 396–408
12. Trost M, Claus T, Herrmann F (2022) Social Sustainability in Production Planning: A Systematic Literature Review. Sustainability 14:8198. https://doi.org/10.3390/su14138198
13. Grabot B, Letouzey A (2000) Short-term manpower management in manufacturing systems: new requirements and DSS prototyping. Comput Ind 43:11–29
14. Attia E-A, Duquenne P, Le-Lann J-M (2014) Considering skills evolutions in multi-skilled workforce allocation with flexible working hours. Int J Prod Res 52:4548–4573. https://doi.org/10.1080/00207543.2013.877613
15. Kagermann H, Lukas W-D, Wahlster W (2011) Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution. VDI-Nachrichten
16. Fraunhofer-Institut für Arbeitswirtschaft und Organisation Stuttgart MyCPS. www.mycpstoolbox.de. Accessed 28 Aug 2021
17. Flüter-Hoffmann C, Hammermann A, Stettes O (2019) Zeitreich – Erfolg mit flexiblen Arbeitszeitmodellen: Leitfaden für Personalverantwortliche und Geschäftsleitungen. Initiative Neue Qualität der Arbeit, Geschäftsstelle c/o Bundesanstalt für Arbeitsschutz und Arbeitsmedizin, Berlin
18. Ernst AT, Jiang H, Krishnamoorthy M, Sier D (2004) Staff scheduling and rostering: A review of applications, methods and models. Eur J Oper Res 153:3–27. https://doi.org/10.1016/S0377-2217(03)00095-X
19. Van den Bergh J, Beliën J, De Bruecker P, et al (2013) Personnel scheduling: A literature review. Eur J Oper Res 226:367–385. https://doi.org/10.1016/j.ejor.2012.11.029
20. Özder EH, Özcan E, Eren T (2020) A Systematic Literature Review for Personnel Scheduling Problems. Int J Inf Technol Decis Mak 19:1695–1735. https://doi.org/10.1142/S0219622020300050
21. Nurmi K, Kyngäs N (2021) A Successful Three-Phase Metaheuristic for the Shift Minimization Personal Task Scheduling Problem. Adv Oper Res 2021:1–12. https://doi.org/10.1155/2021/8876990
22. Kletzander L, Musliu N (2020) Solving the general employee scheduling problem. Comput Oper Res 113:104794: https://doi.org/10.1016/j.cor.2019.104794
23. Krishnamoorthy M, Ernst AT, Baatar D (2012) Algorithms for large scale Shift Minimisation Personnel Task Scheduling Problems. Eur J Oper Res 219:34–48. https://doi.org/10.1016/j.ejor.2011.11.034
24. Sammarco M, Fruggiero F, Neumann WP, Lambiase A (2014) Agent-based modelling of movement rules in DRC systems for volume flexibility: human factors and technical performance. Int J Prod Res 52:633–650. https://doi.org/10.1080/00207543.2013.807952
25. Andrade-Pineda JL, Canca D, Gonzalez-R PL, Calle M (2020) Scheduling a dual-resource flexible job shop with makespan and due date-related criteria. Ann Oper Res 291:5–35. https://doi.org/10.1007/s10479-019-03196-0

26. Thürer M, Zhang H, Stevenson M, et al (2020) Worker assignment in dual resource constrained assembly job shops with worker heterogeneity: an assessment by simulation. Int J Prod Res 58:6336–6349. https://doi.org/10.1080/00207543.2019.1677963
27. Zheng X, Wang L (2016) A knowledge-guided fruit fly optimization algorithm for dual resource constrained flexible job-shop scheduling problem. Int J Prod Res 54:5554–5566. https://doi.org/10.1080/00207543.2016.1170226
28. Agnetis A, Murgia G, Sbrilli S (2014) A job shop scheduling problem with human operators in handicraft production. Int J Prod Res 52:3820–3831. https://doi.org/10.1080/00207543.2013.831220
29. Kress D, Müller D, Nossack J (2019) A worker constrained flexible job shop scheduling problem with sequence-dependent setup times. Spectr 41:179–217. https://doi.org/10.1007/s00291-018-0537-z
30. Qu S, Wang J, Govil S, Leckie JO (2016) Optimized Adaptive Scheduling of a Manufacturing Process System with Multi-skill Workforce and Multiple Machine Types: An Ontology-based, Multi-agent Reinforcement Learning Approach. Procedia CIRP 57:55–60. https://doi.org/10.1016/j.procir.2016.11.011
31. Müller D, Kress D (2021) Filter-and-fan approaches for scheduling flexible job shops under workforce constraints. Int J Prod Res 1–23. https://doi.org/10.1080/00207543.2021.1937745
32. Denkena B, Dittrich MA, Winter F, Wagener C (2016) Simulation-based planning and evaluation of personnel scheduling in knowledge-intensive production systems. Prod Eng 10:489–496. https://doi.org/10.1007/s11740-016-0693-4
33. Altendorfer K, Schober A, Karder J, Beham A (2021) Service level improvement due to worker cross training with stochastic worker absence. Int J Prod Res 59:4416–4433. https://doi.org/10.1080/00207543.2020.1764126
34. Wikarek J, Sitek P (2021) Proactive and reactive approach to employee competence configuration problem in planning and scheduling processes. Appl Intell. https://doi.org/10.1007/s10489-021-02594-x
35. Egilmez G, Erenay B, Süer GA (2014) Stochastic skill-based manpower allocation in a cellular manufacturing system. J Manuf Syst 33:578–588. https://doi.org/10.1016/j.jmsy.2014.05.005
36. Guyon O, Lemaire P, Pinson É, Rivreau D (2014) Solving an integrated job-shop problem with human resource constraints. Ann Oper Res 213:147–171. https://doi.org/10.1007/s10479-012-1132-3
37. Frihat M, B.Hadj-Alouane A, Sadfi C (2022) Optimization of the integrated problem of employee timetabling and job shop scheduling. Comput Oper Res 137:105332. https://doi.org/10.1016/j.cor.2021.105332
38. Bauer W (2015) Selbstorganisierte Kapazitätsflexibilität in Cyber-Physical-Systems: Abschlussbericht. Fraunhofer Verl, Stuttgart
39. Herrmann F (2016) Using Optimization Models for Scheduling in Enterprise Resource Planning Systems. Systems 4:15. https://doi.org/10.3390/systems4010015
40. Baker KR (2013) Computational results for the flowshop tardiness problem. Comput Ind Eng 64:812–816. https://doi.org/10.1016/j.cie.2012.12.018
41. Kramer O (2017) Genetic algorithm essentials. Springer, Cham
42. Fortin F-A, Rainville F-MD, Gardner M-A, et al (2012) DEAP: Evolutionary Algorithms Made Easy. J Mach Learn Res 13:2171–2175
43. SimPy (Version 4.0.1). https://simpy.readthedocs.io/
44. Miller BL, Goldberg DE (1995) Genetic Algorithms, Tournament Selection, and the Effects of Noise. Complex Syst 9:193–212
45. Goldberg DE, Korb B, Deb K (1989) Messy Genetic Algorithms: Motivation, Analysis, and First Results. Coplex Syst 3:493–530
46. Deb K, Agrawal RB (1995) Simulated Binary Crossover for Continuous Search Space. Complex Syst 9:
47. Bonyadi MR, Michalewicz Z (2015) Locating Potentially Disjoint Feasible Regions of a Search Space with a Particle Swarm Optimizer. In: Datta R, Deb K (eds) Evolutionary Constrained Optimization. Springer India, New Delhi, pp 205–230
48. Coello Coello CA (2002) Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Comput Methods Appl Mech Eng 191:1245–1287. https://doi.org/10.1016/S0045-7825(01)00323-1
49. Behnke D, Geiger MJ (2012) Test Instances for the Flexible Job Shop Scheduling Problem with Work Centers. https://doi.org/10.24405/436
50. Brandimarte P (1993) Routing and scheduling in a flexible job shop by tabu search. Ann Oper Res 41:157–183. https://doi.org/10.1007/BF02023073
51. Dauzère-Pérès S, Paulli J (1997) An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. Ann Oper Res 70:281–306. https://doi.org/10.1023/A:1018930406487
52. Fattahi P, Saidi Mehrabad M, Jolai F (2007) Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. J Intell Manuf 18:331–342. https://doi.org/10.1007/s10845-007-0026-8