Journal of Industrial Information Integration 28 (2022) 100350

Contents lists available at ScienceDirect

Journal of Industrial Information Integration

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

Full length article

A hybrid metaheuristic solution approach for the cobot assignment and job shop scheduling problem

Alexander Kinast [a,*], Roland Braune [b], Karl F. Doerner [b], Stefanie Rinderle-Ma [c], Christian Weckenborg [d]

[a] University of Vienna, Forschungsplattform Data Science, Kolingasse 14-16, 1090 Wien, Austria
[b] University of Vienna, Department of Business Decisions and Analytics, Oskar-Morgenstern-Platz 1, 1090 Wien, Austria
[c] Technical University of Munich, Department of Informatics, Chair for Information Systems and Business Process Management, Boltzmannstrasse 3, 85748 Garching, Germany
[d] Technische Universität Braunschweig, Institute of Automotive Management and Industrial Production, Mühlenpfordtstr. 23, 38106 Braunschweig, Germany

ARTICLE INFO

Keywords:
Hybrid genetic algorithm
Job shop scheduling
Biased random-key encoding
Collaborative robots
Variable neighborhood search

ABSTRACT

Nowadays, many manufacturing companies are trying to improve the performance of their processes using available innovative technologies such as collaborative robots (cobots). Cobots are robots with whom no safety distance is necessary. Through cooperation with human workers, they can help increase the production speed of existing workstations. The well-known job shop scheduling problem is, therefore, extended with the addition of a cobot to the workstation assignment. The considered objective is to maximize the normalized sum of production costs and makespan. To solve this problem, we propose a hybrid genetic algorithm with a biased random-key encoding and a variable neighborhood search. The hybrid method combines the exploration aspects of a genetic algorithm with the exploitation abilities of a variable neighborhood search. The developed algorithm is applied to real-world data and artificially generated data. To demonstrate the performance of this algorithm, a constraint programming model is implemented and the results are compared. Additionally, benchmark instances from a related problem from the cobot assignment and assembly line balancing, have been solved. The results from the real-world data show how much the objective function can be improved by the deployment of additional robots. The normalized objective function could be improved by up to 54% when using five additional cobots. As a methodological contribution, the biased random-key encoding is compared with a typical integer-based encoding. A comparison with a dataset from the literature shows that the developed algorithm can compete with state-of-the-art methods on benchmark instances.

1. Introduction

1.1. Overview

In modern industry, fully automated robots are already being frequently used. Robots are able to repeatedly carry out the same static task at a high speed and precision, although they are not suitable for highly flexible production environments. In such production environments, human skills are used to get the desired flexibility.

However, in medium-sized companies, the deployment of robots might be often too expensive, and thus, repetitive tasks are done manually. A cheaper alternative to fully automated robots and pure manual work is human–robot collaborations. Collaborative robots (cobots) differ from traditional robots in the sense that no safety distance is necessary for cobots. In [1], it has been mentioned that if cobots are in direct contact with humans, they move slower than typical robots (around 0.5–1 m/s in comparison to the 1.6 m/s of a human actor). Cobots can do some tasks on their own or in cooperation with a human actor; however, since they move slower than a human, it is assumed that they are also slow in executing tasks on their own as well. However, with its assistance to a human worker, the cooperative production speed is greater than a human worker acting alone. According to their manufacturers, they can do jobs like pick and place, screw driving, injection molding, and many more [1].

An example of a typical cooperative task would be a human actor placing a screw on a workpiece and the cobot screws it in. A cobot is flexible in terms of production and can assist different types of workstations. Since cobots are also mobile, it is assumed that the

* Corresponding author.
E-mail addresses: alexander.kinast@univie.ac.at (A. Kinast), roland.braune@univie.ac.at (R. Braune), karl.doerner@univie.ac.at (K.F. Doerner), stefanie.rinderle-ma@tum.de (S. Rinderle-Ma), c.weckenborg@tu-braunschweig.de (C. Weckenborg).

https://doi.org/10.1016/j.jii.2022.100350
Received 22 March 2021; Received in revised form 26 November 2021; Accepted 21 April 2022
Available online 26 April 2022

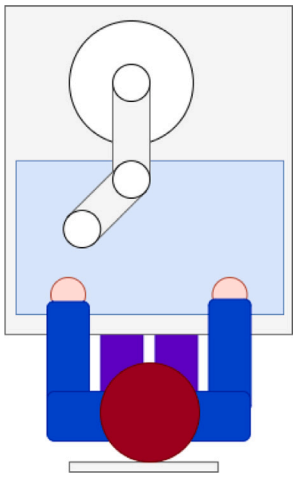

Fig. 1. Human-cobot interaction.

deployment and programming of a cobot can be done within half a day. As the assignment of cobots to workstations is not very time-consuming, the combined problem of cobot assignment and scheduling can be considered an operational problem. A similar kind of motivation can be found in [1].

This innovative form of human–robot interaction is used to increase productivity and/or reduce the number of stressful tasks a human has to carry out. In these human–robot collaborations, a human worker acts closely together with a cobot, often on a single workspace on the same workpiece/task. An example of a raising field in cobot applications is the end-of-life disassembly of electric vehicle batteries. They must be disassembled for recycling purposes and have a high negative impact on the environment if disposed in a wrong way. The disassembly is not easy, since the battery contains substances that are hazardous to humans, and it is important that the cells of the battery are not damaged in the disassembling process. Additionally, these end-of-life disassembling tasks might be of unpredictable volume and high variations due to the difference in car models. Robots are not applicable in such a disassembling process as there is much variation in the types of battery. However, the processes of production and assembly contain many steps that can be done by a cobot. Batteries are held together by many screws, and screwing/unscrewing is a repetitive and uninteresting task for a human. More details on battery disassembly can be found in [2].

However, other examples also exist; in [3], two other electronic devices, a camcorder and a PC have been described. Both products consist of valuable materials, and a cobot could be used to assist a human in different processing steps.

In Fig. 1, it can be seen that the cobot is placed in such a way that it can interact with the human worker in different production processes. By working closely together with a human actor, cobots can reduce the costs and risks involved in this process.

Typically, in these production systems, the tasks need to be assigned to specific workstations. As the assignment of cobots is not very time-consuming, the assignment of cobots to workstations is considered an operative problem. These cobots can be used to speed up bottleneck workstations in a production process. Cobots could also be beneficial in classical manufacturing processes. In the numerical study, a real-world problem where engines, casings, and other machine parts are produced is considered.

1.2. Related work and research contribution

A lot of research has been done in the field of job shop scheduling since the late 1950s. Researchers have tried to optimize the problem with different well-known performance measures or objective functions such as makespan, mean flow time (the average time a single job spends in the shop), or lateness of the jobs (how well due dates are met). Various heuristics such as tabu search, genetic algorithms, and variable neighborhood search have been used to solve the problem with single or multiple objectives. In [4], an in-depth review of job shop scheduling solution strategies has been done. A total of 62 papers have been reviewed, and most of these papers are research papers that focus on method development. Only around 8% address real-world industrial applications.

In [5], an overview of the state-of-the-art research in the field of genetic algorithms for the flexible job shop scheduling problem is given. In this paper, 190 publications from the year 2001 to 2017 have been reviewed. The two major research areas are genetic algorithms (with 79 publications) and hybrid genetic algorithms (with 75 publications).

In [6], a genetic algorithm is combined with a variable neighborhood search to solve different deterministic benchmark instances for the flexible job shop scheduling problem. The results show that the hybrid genetic algorithm is a state-of-the-art algorithm that performs remarkably on benchmark instances.

Another research direction focuses on the influence of disruptions and rescheduling of processes. Examples of such disruptions are machine breakdowns, operator illness, new priority jobs, canceled jobs, or changes in job deadlines. There are various methods of how to react to such disruptions. One option would be full-reactive scheduling with priority rules. This means that the decision of which task is to be performed next is done locally on the machine. Each task gets assigned a priority based on machine and job attributes. Another option would be to schedule jobs in a more robust way so that even with machine breakdowns, the objective function is influenced to the smallest extent possible [7].

A different way to deal with uncertainties in production is the application of fuzzy sets. They can be used to model the uncertainties in a process such as uncertain processing time and uncertain due dates. Based on these fuzzy sets, a satisfaction grade for different objective values can be calculated. Meta-heuristics such as a genetic algorithm can then be applied for the optimization of the satisfaction grade [8].

Some researchers even propose to apply deep reinforcement learning for job shop scheduling problems. Therefore, an agent-based learning approach could be used. This approach has a state that describes the current situation of the environment and has actions that it can take. When an action is taken, a positive or negative reward is received. This is known as a short-term reward. It is also possible to add a so-called Q-value that considers the long-term rewards of an action. Results that can be achieved with these deep reinforcement learning approaches are nowhere near what can be achieved with metaheuristics [9].

This paper is an extended version of [10] and the research contribution of this paper is threefold. Based on the original paper, a new method, the hybrid genetic algorithm has been developed here and the numerical study has been expanded.

In Table 1, a comparison with the most related paper in the literature [1] is given. In this paper, the combined cobot assignment and assembly line balancing problem was first introduced. The research contribution of this paper is summarized in Table 1 and is:

- In the first entry of Table 1, it can be seen that the first research contribution of this paper is the job shop scheduling problem being extended with a cobot assignment problem.
- In Table 1 in the second and third entry, the second research contribution can be seen. To solve the newly introduced problem, we combine the exploratory strength of a genetic algorithm with the exploitative aspects of a variable neighborhood search. A new and alternative biased random-key encoding, developed in [11] is used and a performance improvement compared to the standard integer encoding is shown. The efficiency of the algorithm is also shown by comparing it to a developed constraint programming

Table 1
Literature comparison.

No.	Category	Weckenborg et al.	Kinast et al.
1.	Solved Problem	"Combined cobot assignment and assembly line balancing problem"	"Combined cobot assignment and assembly line balancing problem" and "Combined cobot assignment and job shop scheduling problem"
2	Algorithms	Mixed integer programming and genetic algorithm	Constraint programming and genetic algorithm with variable neighborhood search
3	Genetic algorithm encoding	Integer-based encoding	Integer-based and biased random-key encoded
4	Data set	Generated data sets for the simple assembly line balancing problem with cobot assignment with 20, 50, and 100 tasks.	Real-world based job shop scheduling problem with cobot assignment with up to 1265 tasks that have to be scheduled to 54 workstations. Generated data sets with up to 1200 tasks.

(CP) model and by solving benchmark instances from the literature for the related cobot assignment and assembly line balancing problem.
- In Table 1 in the fourth entry, the size of the solved instances is shown. The third contribution is the managerial insights of the savings when different numbers of cobots are used for the combined cobot assignment and job shop scheduling problem for real-world instances.

2. Problem description

2.1. Cobot assignment and job shop scheduling

Many companies nowadays already know what orders have to be produced over the next weeks or even months. Preparing sufficient resources in order to handle all incoming orders is a tactical problem as traditional resources such as workstations or employees need long preparation times. Workstations need to be produced and installed, while workers need to be employed and trained. Typically, these resources do not change after a planning period ends. In one planning period, all existing orders need to be assigned to the given resources in a way that a given objective function is minimized.

This traditional resource planning problem cannot be applied if a company has invested in innovative technology such as cobots. Since cobots have short setup times, they can be deployed to a bottleneck workstation before a new planning period starts. To fully utilize the possibilities that such technologies can offer, the classical job shop scheduling problem has to be combined with a cobot to workstation assignment. This combined approach of cobot assignment and job shop scheduling problem was first introduced in [10].

A typical job shop scheduling problem consists of jobs, tasks, and workstations. A job consists of a chain of tasks that must be processed in a given order on specific workstations or on any workstation (simplified job shop scheduling problem). This given order is modeled in a precedence graph. A task is a production step that is necessary for the completion of a whole job, for example, mounting of a mechanical part or screwing in screws. A precedence relationship could be that a raw material needs to be formed in a melting furnace before it can be further processed. For each task, a measured standard production time exists. In a deterministic version, it is assumed that this standard production time is the time that a task needs to be completed.

In scheduling problems, it is often the case that a company has multiple similar workstations that can do the same production task. A workstation can be either a machine or a station with human actors. Depending on the type of the machine or the number of workers, the speed and production cost of such a workstation can vary. Workstations that can do the same tasks are grouped together as workstation groups. We consider the workstations within a workstation group as heterogeneous. In particular, when a cobot is assigned to a workstation within a workstation group, the processing times of tasks on this specific workstation decreases. To find the best suiting workstation in a workstation group for a task in order to improve the objective function, the classical job shop scheduling problem is extended with the workstation assignment task, as already introduced in [12].

In our combined cobot assignment and scheduling problem, the following decisions have to be made:

- As only a limited number of cobots can be bought at a time, at which workstations should they be installed in a given planning period?
- If tasks can be produced on multiple workstations, on which workstation should a task be produced?
- If multiple tasks can be produced at the same time on a workstation, which task should be prioritized?

Different objective functions, which optimize this combined cobot assignment and job shop scheduling problem, can be used. An example would be to minimize the production costs, which would result in the highest profit for a predetermined set of orders. Only considering profits or costs will not be applicable for a real-world production, since the cheapest workstation in a group will then have more tasks assigned, while the rest will be neglected. In real productions, a delayed delivery of the ordered jobs will often have various different negative consequences. To prevent such consequences, a second objective function would be to minimize the makespan. However, this will lead to high production costs since, regardless of the production costs, all workstations should keep producing in parallel. To find a good compromise, the appropriate objective function will be a combination of these two objective functions. Therefore, we normalize the production cost and the makespan and factor them equally in the objective function. There might be solutions where the reduced production cost outweighs an increased makespan.

In [13], it is described how multiple objective values can be normalized to have the same influence on the objective function. For each objective function, a minimum F_{min} and maximum F_{max} value has to be found. Based on this maximum and minimum, a range for the valid values is calculated. With the following formula, the normalized value is closer to 1 if the value is closer to the maximum:

$$N = \frac{value - F_{min}}{F_{max} - F_{min}}$$

With this formula, the normalized value is closer to 1 if the value is closer to the minimum:

$$N = \frac{F_{max} - value}{F_{max} - F_{min}}$$

Depending on the objective function, it might be better to produce either on a faster workstation or on a workstation with less production costs. This means the choice of the objective function will influence on what workstations products are produced and where cobots are installed. The fitness F of our general weighted objective function, that is a combination of the normalized production cost n_{cost} and the normalized makespan $n_{makespan}$, can be described as follows:

$$F = n_{cost} + n_{makespan}$$

A detailed problem description is given based on the simple example provided in Fig. 2. The main interest of this paper is to solve a combined cobot assignment and scheduling problem motivated by a real-world problem introduced by our industry partner. An example order from

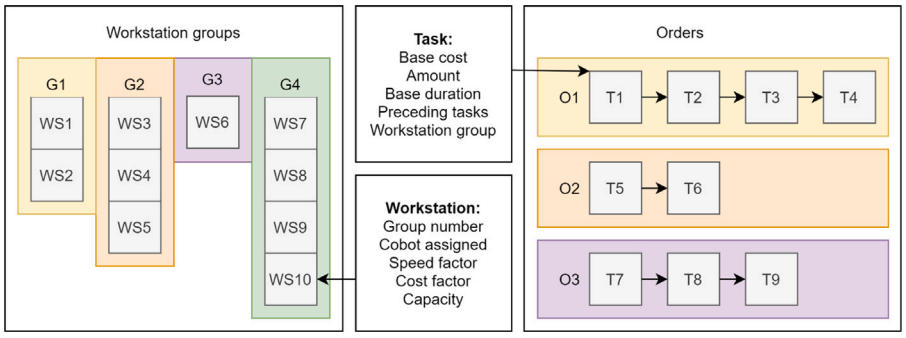

Fig. 2. Real-world problem overview.

the real-world dataset is the production of housing. Multiple tasks have to be executed on base materials such as deburring, drilling, milling, cutting a thread, checking the drilling, and packing. On the right-hand side of Fig. 2, it is illustrated that the data contains orders (O1, O2, O3) that group tasks (T1, ..., T9) together. Tasks in that order have a fixed sequence of production. The arrows indicate the precedence relation. However, it is possible to produce tasks of other orders in between two tasks of another order, as tasks preemption is not allowed and tasks have a standard production time and costs (base duration and base costs, respectively). For each task, the preceding tasks are known and the required workstation group for the specific tasks are given. Each workstation has a specific cost and speed factor. With these factors and the standard production time and speed factor, the real production time and real costs are calculated for each task assigned to a workstation. Workstations that can process similar tasks are grouped into workstation groups. These workstation groups can be seen on the left-hand side of Fig. 2. Each workstation in such a group (G1, G2, G3, G4) has individual production costs and production speed, given by speed and cost factor, respectively. Based on the number of produced products for a workstation in use, there will be setup, de-setup, and production costs and times. The production costs and speed can vary across several workstations in one workstation group. We assume that cobots can be installed on all workstations. Furthermore, we also assume that the tasks can be produced on all workstations of that specific workstations group. In some real-world problem settings, it can be the case that some specific tasks have a fixed workstation given. For the workstations, the following different capacity modes exist:

- One product at a time (typical assembly workstation)
- Space capacity (e.g., an oven)
- Unlimited (assumed if the task is done externally)

2.2. Cobot assignment and assembly line balancing

In order to evaluate the performance of our developed algorithm, we adapted our method such that it can be applicable to a related problem already existing in the literature [1]. This related problem is an assembly line balancing problem that has been extended with the help of a cobot to a workstation assignment. In this problem, tasks can be executed by the human worker, in collaboration of worker and cobot, or by an individual cobot without human assistance.

In [1], a problem formulation of generalized assembly line balancing is proposed, which extends existing literature regarding the cobot to workstation assignment. Benchmark instances provided by [14] are adapted to cover for the extended scope of the problem. In Fig. 3, a typical example of one instance of a problem in the literature has been shown. On the left-hand side, general settings such as the number of workstations or robots can be seen. In the middle, all tasks and the production times of the tasks can be seen (task number and task execution time for human, robot, or the collaborative execution). The robot flexibility (RF) and cooperative flexibility (CF) in a general setting describe the share of tasks that can be done by the cobot or collaboratively. In this simple example, 40% (8) of the tasks can be produced by the cobot, 40% (8) of the tasks can be produced collaboratively, and all tasks can be executed by a human alone. It is not necessary that a task that can be produced collaboratively should be able to be produced by a robot and vice versa. On the right-hand side, the precedence graph is shown. To start a task, all preceding tasks have to be completed either on the current workstation or a previous workstation.

Table 2
Indices used in the CP formulation.

o	Order
i	Task
j	Task slice (relative to task)
w	Workstation
k	Machine (relative to workstation)

Table 3
Parameters used in the CP formulation.

n_i	Number of slices for task i.
m_w	Number of machines on workstation w.
b	Number of available cobots.
γ_w	Speed factor of workstation w.
δ_w	Cost factor of workstation w.
φ	Cobot acceleration factor.
$\mathcal{O}$	Set of all orders.
$\mathcal{I}$	Set of all task indices.
$\mathcal{I}^o$	Set of task indices included in order o.
$\mathcal{W}$	Set of all workstations.
$\mathcal{W}^i$	Set of all workstations on which task i can be produced.
$\mathcal{J}^w$	Set of pairs (i, j) (task i, slice j) that can be assigned to workstation w.
$\theta(i)$	Function yielding the order index of task i.
p_i	Production/processing time of task i.
$Y_w(o, o')$	Sequence-dependent setup time when changing from order ID o to o' on workstation w.

3. A constraint programming formulation of the job shop scheduling problem with cobot assignment

In this section, we present a Constraint Programming formulation for the scheduling problem stated in Section 2.1. From a structural point of view, the model aims at scheduling separate *slices* of tasks. Each slice constitutes a piece of work in the given production context and all pieces within a task are of the same (material) type. Let $\mathcal{O}$ denote the set of all orders and $\mathcal{I}$ the set of all tasks. It is assumed that the tasks are numbered consecutively across orders. The processing or production time of a task $i \in \mathcal{I}$ is denoted by p_i and applies to each slice j of a task. To simplify the notation, a slice j of a task i is also denoted by the pair (i, j) henceforth.

Let $\mathcal{W}$ be the set of all workstations. The number of (parallel) machines on a workstation $w \in \mathcal{W}$ is denoted by m_w. Each task i is

General settings	
Stations:	5
Robots	1
RF:	0.4
CF:	0.4

Task times			
Task number	Human	Robot	Collaborative
1	100	200	70
2	300	600	∞
3	200	∞	140
⋮			
19	400	800	∞
20	150	∞	∞

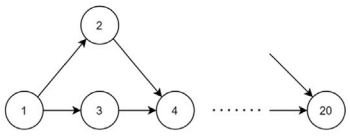

Fig. 3. Simple assembly line balancing problem overview.

Table 4
Interval variables used in the CP model formulation.

Symbol	Optional	Size	Description
U_{ij}	no		Represents the production time of slice j of task i.
V_{ij}^{wk}	yes	$p_i \cdot \gamma_w$	Optional interval for execution of task slice (i,j) on machine k of workstation w.
$\hat{V}_{ij}^{wk}$	yes	$p_i \cdot \gamma_w \cdot \varphi$	Optional interval for *cobot-assisted* execution of task slice (i,j) on machine k of workstation w.
D'_{wk}	no	0	Dummy start interval for machine k on workstation w.
D''_{wk}	no	0	Dummy end interval for machine k on workstation w.
B_w	yes		Cobot master interval for workstation w.

Table 5
Sequence variable definition, one for each machine k of workstation w.

Symbol	Interval Var. Set	Setup Type
Ψ_{wk}	$\{V_{ij}^{wk} \mid (i,j) \in \mathcal{J}^w\} \cup$ $\{\hat{V}_{ij}^{wk} \mid (i,j) \in \mathcal{J}^w\} \cup$ $\{D'_{wk} \mid w \in \mathcal{W}, 1 \le k \le m_w\} \cup$ $\{D''_{wk} \mid w \in \mathcal{W}, 1 \le k \le m_w\}$	$(\underbrace{1,\dots,1}_{\sum_{i\in\lvert\mathcal{I}^1\rvert} n_i}, \underbrace{2,\dots,2}_{\sum_{i\in\lvert\mathcal{I}^2\rvert} n_i}, \dots, \lvert\mathcal{O}\rvert, \dots, \lvert\mathcal{O}\rvert,$ $\underbrace{\lvert\mathcal{O}\rvert+1,\dots,\lvert\mathcal{O}\rvert+1}_{\sum_{i\in\lvert\mathcal{I}^1\rvert} n_i}, \dots,$ $\underbrace{2\cdot\lvert\mathcal{O}\rvert+1,\dots,2\cdot\lvert\mathcal{O}\rvert+1}_{2\cdot\sum_w m_w})$

assigned a set $\mathcal{W}^i$ of *eligible* workstations on which it can be processed. Each workstation has a speed factor γ_w and a cost factor δ_w both of which are constant for all the machines on the workstation. Each slice of a task requires exactly one machine at a time and the slices of a task can be processed sequentially on a single machine, in parallel on multiple machines, or in a mixed fashion.

If two slices of different orders are processed consecutively on the same machine, the machine configuration has to be changed. First, some work has to be done for the task that is processed earlier (the leaving task) and after that, the machine has to be prepared for the next (entering) task. These activities are referred to as de-setup (teardown) and setup and covered in the CP formulation by means of *sequence-dependent* setup times. In fact, the two activities, de-setup and setup, are combined into a single one. The time required for this activity is given by $Y_w(o,o')$, depending on the workstation w, and the orders o and o' to which the slices involved in the transition (from o to o') belong to.

The CP model formulation is made up of different kinds of interval variables as summarized in Table 4. Variables V_{ij}^{wk} and $\hat{V}_{ij}^{wk}$ are the core variables, reflecting the different execution modes that are available for a slice j of task i. There is one optional interval for each workstation w and machine k on which the task can be executed, each of which is set to a fixed size, that is, the production time p_i multiplied with the workstation's speed factor γ_w. The counterparts $\hat{V}_{ij}^{wk}$ reflect the corresponding cobot-assisted execution modes. To control the cobot assignment itself, interval variables B_w indicate the presence of a cobot at a particular workstation w. Variables U_{ij} are for structural purposes only, to make sure that exactly one of the optional intervals V_{ij}^{wk} and $\hat{V}_{ij}^{wk}$ is chosen for a particular slice (i,j).

The machines on a workstation are renewable resources with unary capacity and thus require a disjunctive scheduling approach. For this purpose, the formulation relies on sequence variables Ψ_{wk}, with $w \in \mathcal{W}$ and $1 \le k \le m_w$. The definition of the sequence variables Ψ given in Table 5 follows the scheme imposed by IBM ILOG CP Optimizer, but the concept can be transferred to other scheduling-related CP frameworks as well. To consider the sequence-dependent setup times, the definition of the sequence variables requires information on the setup type of each interval variable that can be part of the sequence. Assuming that interval variables are sorted according to task (and thus order) indices in ascending order, the setup type for all slices that belong to the first order is therefore 1, and for all slices of the second order, 2, and so on. Note that distinct setup types have to be used for variables $\hat{V}$, because the cobot-assisted execution also allows to speed up the setup and de-setup activities. To achieve this, fictitious order indices $|\mathcal{O}|+1, |\mathcal{O}|+2, \dots, 2\cdot|\mathcal{O}|$ are assigned to the slices represented by the $\hat{V}$ intervals. To enforce initial setup and final teardown activities, that is, before the first and after the last scheduled slice on a machine respectively, two dummy interval variables of size 0 are added to each machine's sequence. The setup type of these dummy variables is set to $2\cdot|\mathcal{O}|+1$. It must be remarked that functions Y_w, essentially modeling machine-specific setup matrices, are capable of taking arguments from the extended range of order indices described above.

$$C_{\max} = \max_{w\in\mathcal{W}, 1\le k\le m_w} \text{endOf}(D''_{wk}). \tag{1}$$

$$\begin{aligned} TC = &\sum_{w\in\mathcal{W}} \sum_{(i,j)\in\mathcal{J}^w} \sum_{1\le k\le m_w} \text{presenceOf}(V_{ij}^{wk}) \cdot p_i \cdot \gamma_w \cdot \delta_w \\ &+ \sum_{w\in\mathcal{W}} \sum_{(i,j)\in\mathcal{J}^w} \sum_{1\le k\le m_w} \text{presenceOf}(\hat{V}_{ij}^{wk}) \cdot p_i \cdot \gamma_w \cdot \varphi \cdot \delta_w \\ &+ \sum_{w\in\mathcal{W}} \sum_{(i,j)\in\mathcal{J}^w} \sum_{1\le k\le m_w} \delta_w \\ &\quad \cdot (Y_w(\theta(i), \text{typeOfNext}(\Psi_{wk}, V_{ij}^{wk})) \\ &+ Y_w(\theta(i), \text{typeOfNext}(\Psi_{wk}, \hat{V}_{ij}^{wk}))) \end{aligned} \tag{2}$$

We can now state the formulation itself, based on indices and parameters summarized in Tables 2 and 3, and the interval and sequence

variables from Tables 4 and 5. As for the notation used in the tables, it must be remarked that the model statement also uses the nomenclature of IBM ILOG CP Optimizer but is still general enough to be realized within other CP frameworks with dedicated support for scheduling problems.

Eqs. (1) and (2) formally specify the two objective functions, namely the makespan and the total production cost, as introduced in Section 2.1. The makespan can simply be computed from the maximum finish times of the terminal dummy interval variables on each machine. The production cost is the sum of all actually allocated machine times (processing/production, setup and de-setup) multiplied by the workstation-specific cost factors δ_w.

$$\text{Maximize} \quad \frac{\overline{C_{\max}} - C_{\max}}{\overline{C_{\max}} - \underline{C_{\max}}} + \frac{\overline{TC} - TC}{\overline{TC} - \underline{TC}} \tag{3}$$

subject to

$$\text{alternative}(U_{ij}, \{V_{ij}^{wk} \mid w \in \mathcal{W}^i, 1 \le k \le m_w\} \cup \{\hat{V}_{ij}^{wk} \mid w \in \mathcal{W}^i, 1 \le k \le m_w\}) \quad \forall i \in \mathcal{I}, \forall 1 \le j \le n_i, \tag{4}$$

$$\text{startOf}(U_{i,j}) \le \text{startOf}(U_{i,j+1}) \quad \forall i \in \mathcal{I}, \forall 1 \le j < n_i, \tag{5}$$

$$\text{endBeforeStart}(U_{i,n_i}, U_{i+1,1}) \quad \forall o \in \mathcal{O}, \forall i \in \mathcal{I}^o, i < |\mathcal{I}^o|, \tag{6}$$

$$\text{noOverlap}(\Psi_{wk}, Y_w, 1) \quad \forall w \in \mathcal{W}, \forall 1 \le k \le m_w, \tag{7}$$

$$\text{span}(B_w, \{\hat{V}_{ij}^{wk} \mid 1 \le k \le m_w, (i,j) \in \mathcal{J}^w\}) \quad \forall w \in \mathcal{W}, \tag{8}$$

$$\sum_{w \in \mathcal{W}} \text{presenceOf}(B_w) = b. \tag{9}$$

The actual objective function used in the formulation is then given by the sum of the normalized makespan and cost values, as can be seen from Eq. (3). The normalization is based on minimum and maximum values for each component objective, that is $\underline{C_{\max}}$ and $\overline{C_{\max}}$ for the makespan, and $\underline{TC}$ and $\overline{TC}$ for the total production cost.

Constraints (4) ensure that exactly one execution mode is chosen for each slice (i, j). To reduce the symmetry, constraints (5) impose a partial order between slices of the same task, still allowing that two or more slices can be scheduled in parallel. The precedence among tasks of the same order is accomplished through constraints (6), by simply introducing end-before-start requirements between the last and the first slice of two subsequent tasks. The no-overlap constraints (7) are responsible for disjunctive scheduling on the machines, also considering the sequence-dependent setup times. The span constraints (8) enforce the presence of a cobot master interval B_w as soon as at least one slice is scheduled in a cobot-assisted execution mode on any machine of workstation w. Constraints (9) finally limit the cobot usage by placing an upper bound on the number of interval variables B_w that can be present.

4. Solution method

4.1. Overview

In [15], it has been described that the job shop scheduling problem is an NP-hard problem. As described in Section 1.2, various metaheuristics such as genetic algorithms or constraint programming approaches have been applied successfully on large instances in the literature. Metaheuristics are used to receive approximations to the global optimum of the problem-specific objective function [16]. As described in the previous chapters, in this study, we have extended the job shop scheduling problem with a cobot to workstation assignment. This means that the number of decisions has increased. Therefore, it is necessary to use state-of-the-art metaheuristics to solve this problem.

In recent years, the trend has been to combine different heuristics to so-called hybrid metaheuristics to exploit the strengths of different metaheuristic search concepts. This research direction is pushed by the fact that hybrid metaheuristics often outperform traditional heuristic or metaheuristic approaches on hard optimization problems. By combining the strengths of the individual algorithms, they are able to work together, resulting in a synergy that can outperform individual methods. An example of such a well-known hybrid metaheuristic is the combination of population-based methods with local search methods. In such combinations, the exploratory nature of population-based methods is combined with a local search on promising regions [17].

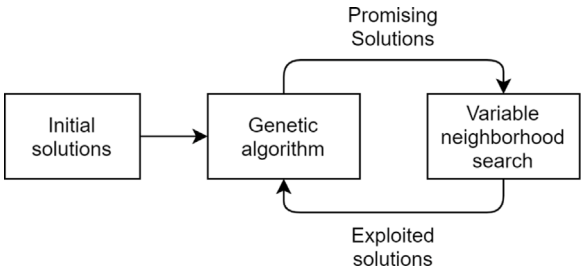

Fig. 4. Overview hybrid genetic algorithm.

In Fig. 4, an overview of such a hybrid genetic algorithm has been provided. The general idea of our algorithm is that promising solutions of the genetic algorithm are improved by local search-based methods. These improved solutions replace the original non-local optimized solutions of the genetic algorithm. In [18], it is described that hybrid genetic algorithms are also called memetic algorithms.

These memetic algorithms have been applied successfully to many optimization problems, including classical job shop scheduling problems. In many cases, they outperform traditional algorithms by using key features of several algorithms [19]. The intention is to combine the exploratory features of the genetic algorithm with the capability of the variable neighborhood search to intensify the search within the promising regions of the search space.

Algorithm 1
Pseudo code genetic algorithm.

0	Initialize	Initialize the population with random individuals
1	Evaluation	Evaluate all individuals in the current generation
2	while(!termination)	While termination criteria not reached
3	Selection	Select parents for the new generation
4	Crossover	Create children out of the parents
5	Mutation	Mutate children based on a given probability
6	Create Generation	Create a new generation based on the created children
7	Evaluation	Evaluate all individuals in the new generation
8	end while	

4.2. Genetic algorithm

In Algorithm 1, the pseudocode for a generic genetic algorithm is given. In line 0, it can be seen that the genetic algorithm starts by creating a randomly initialized population. In line 1, it can be seen that a fitness value is assigned to each individual in the initial generation. In Algorithm 1 in line 2, it can be seen that this randomly generated population is improved over the duration of the algorithm until a stopping criterion is reached. Examples are a maximum number of generations, a time limit, or the finding of an acceptable solution.

To generate new solutions, the algorithm uses the steps from Algorithm 1 in lines 3 to 7 until a stopping criterion is reached:

- **Selection:** Selecting individuals from the current generation that act as parents for the next generation. Fitter individuals have a higher chance of being selected as a parent.
- **Crossover:** Taking two parents to create one or two new solutions for the next generation. Different crossover variations exist.

- **Mutation:** Changing a solution such that new points in the solution space are discovered. This should prevent premature convergence of the algorithm.
- **Create Generation:** The generated solutions are used to create the new generation in the genetic algorithm.
- **Evaluation:** Assigning a fitness value to every individual of the current generation. The fitness value describes the quality of a solution regarding the selected objective function.

The solution needs to be encoded such that the operators can work with the representation. This means the operators have to be implemented for each representation [20].

To get comparable results for an encoding of an individual, the operators that are used in the genetic algorithm should be comparable between different encodings. The following operators can be implemented for integer and real value encoding and have been used in this contribution:

- **Fitness proportional selection:** The chance of an individuum to become a parent in the next generation is proportional to its fitness. This means fitter individuals have a higher chance of mating.
- **Uniform some positions arithmetic crossover:** Based on a probability, each position of the solution is crossed between two parents. A parameter α defines how close the solution is to either parent one or parent two. For the integer-encoded solution, the rounded version is used.
- **All-positions manipulator:** All positions of the vector are manipulated with a given strength that is defined by a parameter α. For the integer encoded solution, a rounded version is used.

During the development of the genetic algorithm, other operators such as a one-position manipulator and a single point crossover were tested. However, by using these operators, the genetic diversity got lost after some generations, and the results were significantly worse than those generated with the operators described above.

4.2.1. Encoding and evaluation - real-world problem

In the real-world problem, each task can be produced on a group of workstations. For all tasks that are produced on a workstation group with more than one workstation, the workstation is encoded by a double value. This double value is decoded during the evaluation. This can be seen in the gray fields in Fig. 5.

The second value encoded for each task is a priority parameter. If multiple tasks can be produced at a specific workstation, the task with the highest priority is produced first. The priority can be seen in the red fields in Fig. 5.

The last part that has to be encoded is the cobot assignment. This is similar to the workstation encoding of the tasks. In the encoding, a double value is used. This value will be decoded based on all the available workstations that have no cobot assigned yet. An example would be the case where there are 10 workstations to which no cobot has been assigned. For each workstation, a biased random-key encoded cobot would have a value within the range of 0.1. This means that to assign a cobot to workstation 1, the value has to be between 0 (included) and 0.1 (excluded). Thus, the first yellow-encoded value in Fig. 5 would mean that a cobot is assigned to workstation 1.

The biased random-key encoding is compared to a normal job shop scheduling encoding where the tasks for workstations are encoded as integer numbers with bounds depending on the number of available workstations in the workstation group. If task 1 can be produced on workstations 1 to 5, only integer numbers in this range will be generated. Additionally, the priority and the cobot location are also encoded as integer values. This means that all selection, mutation, and crossover operators that can handle an integer array can be used to generate new values for this encoding.

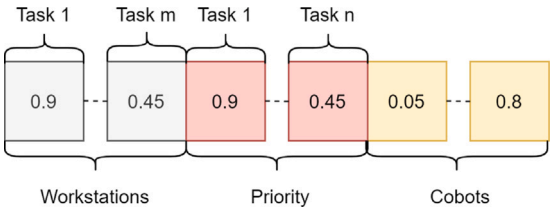

Fig. 5. Biased random-key encoding.

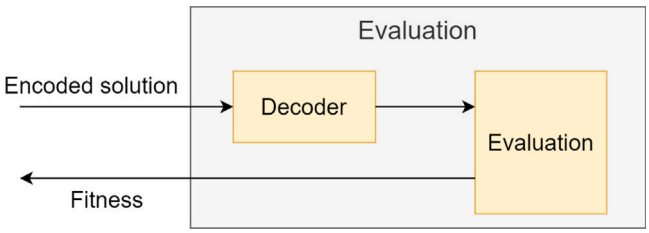

Fig. 6. Decoding and evaluation.

To evaluate a solution, the first step is to assign the available cobots to the workstations. Each workstation has a property speed factor, cost factor, capacity, and a property that defines if a cobot is assigned. If a cobot is assigned to a workstation, it is assumed that the production speed is increased by 30% [1]. This value is realistic for existing cobots, but it can be also exchanged for different values. Additionally, each workstation has a speed and cost factor that is applied to every task produced on a specific workstation. The number of tasks that can be carried out in parallel depends on the capacity of the workstation.

In the second step, all tasks are assigned to the workstations based on the encoded value. Each workstation with remaining capacity checks whether there are tasks to execute. All producible tasks (whose preceding tasks have been finished) are sorted by priority, and the task with the highest priority is produced next. When a task starts producing, its finishing time is calculated based on the task duration, the speed factor of the workstation, and whether a cobot is assigned to the workstation. The cost for producing a task is based on the production cost of the task and the cost factor of the workstation.

Some workstations have setup and de-setup times with a workstation-specific cost factor. Every time a new task is produced on the workstation, the resulting costs will be added to the objective function/fitness function. If a workstation has a capacity, its costs are added only once every time a new product of this task is generated. Since a cobot might be able to assist a human worker in the setup process, it is assumed that the setup speed also increases by 30%.

The objective function that is used to receive a fitness is currently a combination of the normalized production cost and the normalized makespan. The production cost is measured in cent, while the makespan is measured in seconds.

$$F = n_{\text{cost}} + n_{\text{makespan}}$$

Every time the genetic algorithm creates a new encoded solution (initialization and for every individual created using genetic operators), the evaluation method described above is used to create a fitness value. In Fig. 6, we can see that the encoded solution gets decoded and passed to the evaluation method. After the decoding described above, our tasks have all the properties from Fig. 2. Additionally, our tasks have received a priority and a workstation where the tasks are being produced. For all workstations, the properties from Fig. 2 and an additional property, if a cobot has been assigned, are set. In the evaluation, it is checked whether there are any free workstations (workstations with residual capacity) where tasks without non-produced preceding tasks are waiting. If there are tasks waiting, they are ordered by priority

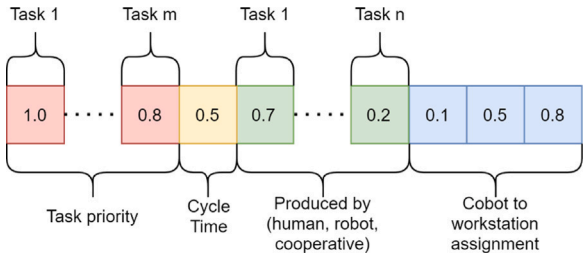

Fig. 7. Biased random-key encoding - Literature data set.

and then assigned to the workstations. During production, the fitness is increased by the base cost of a task multiplied by the cost factor of the workstation. At the end of the evaluation, the fitness is increased by the makespan multiplied by the defined factor.

Since all the steps in the evaluation are deterministic, the evaluation method is deterministic and returns a fitness value to the optimization algorithm.

4.2.2. Encoding and evaluation - assembly line balancing problem with cobot assignment

To encode a solution of the literature problem, a similar approach to the real-world problem biased random-key encoding is used. However, the following important differences between the datasets have to be considered:

- Each task can be produced on each workstation, not only on workstations of a specified workstation group.
- The literature dataset is an assembly line balancing problem of type two, which means that the fitness of one encoded solution is equal to the cycle time.
- Instead of speeding up a workstation by 30%, there are three different production modes. All tasks can be done by a human, and a part of the tasks could be done by the robot (200% of the human production time), whereas a part of the tasks can be done cooperatively at 70% of the human production time.

The encoding used for the problem from the literature has been shown in Fig. 7. Similar to the real-world problem, the cobot to workstation assignment is encoded as biased random-key. Additionally, the red-encoded values are used to assign a priority to all tasks, and tasks with multiple production modes (human, robot, and cooperative) are assigned a "produced by" value. This is the green value in the picture. Since the makespan is used as the fitness value, one additional cycle time value is encoded.

In Fig. 8, the evaluation of one encoded solution is shown. In the first step, cobots are assigned to all the available workstations without cobots based on a biased random-key. If five workstations are available, each workstation has a range of 0.2. This means the biased random-key 0.1 would match the first workstation.

In Fig. 8, in step 2, a maximum cycle time is set for all workstations. This cycle time limit can be estimated with the human production time of all tasks and the number of workstations. The first important value to be calculated is as follows:

$$human\_production\_time\_per\_workstation = \frac{sum\_of\_human\_production\_times}{number\_of\_workstations}$$

Without cobots, this corresponds to the lower bound on cycle time assuming a perfectly smooth allocation of tasks to workstations. In practice, however, a perfectly smooth allocation of tasks to stations cannot necessarily be achieved, as it assumes the divisibility of tasks. Please consider an example with 100 tasks, 10 workstations, and an average human production time of 20 time units. In this example, we would have a human production time per workstation of 200 (100*20/10). If one individual task, however, comprises a duration of more than 200 time units, this cycle time cannot be achieved. Therefore, the second important value is the human production time of the longest individual tasks. For further discussion of the bounds on the cycle time in assembly line balancing problems, please refer to [21].

The encoded cycle time value is now decoded in the following way:

$$max = \text{Max}(human\_production\_time\_per\_workstation, max\_individual\_task)$$

$$cycle\_time = max \cdot (0.8 + encoded\_cycle\_time \cdot 0.4)$$

The base value of this formula is the maximum of the average human production time per workstation and the longest individual task. Based on the encoded cycle time, the cycle time limit for all workstations is between 80% and 120% of this value. In Fig. 8, this can be seen after step 2.

In Fig. 8, in step 3, we can see how tasks are assigned to the workstations. For each empty workstation, the following steps are repeated until no more tasks can be assigned:

- Get the list of producible tasks (all preceding tasks have been finished);
- Find the next task that can be assigned to this workstation based on the given task priority and the assigned production mode;

In Fig. 8, after step 3, we can see what a typical solution would look like. The different production modes are colored (human: red; robot: yellow; cooperative: violet), and the upper and lower parts of a workstation symbolize the time of the worker and the robot, respectively. Additionally, it can be seen that the real fitness of this individual might be different than the calculated cycle time limit of the workstations. If it is not possible to assign all tasks to the workstations because the cycle time limit is too low, a penalty value is assigned to this solution.

4.3. Variable neighborhood search

A variable neighborhood search is a metaheuristic that uses a local search method to systematically search neighborhoods with increasing distances [22]. Since we only use the variable neighborhood search in the hybrid genetic algorithm and not as an individual metaheuristic, the encoding of the genetic algorithm is used. This encoding is problem-dependent and has been described in detail in the previous two sections. The neighborhood $N_k(x)$ can be defined as all the solutions that can be reached with k changes from a starting solution x. Typical local search heuristics usually use $k = 1$. Since the variable neighborhood search is only used in combination with the genetic algorithm, the neighborhood of a solution is described in the next chapter in the context of the hybrid genetic algorithm. In Algorithm 2, the basic steps of a variable neighborhood search are explained. To start the metaheuristic, k_{max} is considered the maximum distance to neighboring solutions and a termination criterion for how long each neighborhood is searched. Examples of this termination criterion are the maximum number of evaluated solutions or a time limit. After initializing these variables, the main loop starts. In this main loop, neighboring solutions are searched and evaluated until the termination criterion is reached (e.g., evaluation of 20 neighboring solutions). In Algorithm 2, from line 4 to line 7, we can see a first improvement strategy. This means if any of the generated solutions x' is better than the current best solution x, the current x is replaced by x' and the variable neighborhood search starts with $k = 1$ at the new solution. If no better solution is found in the current neighborhood until the termination criterion is reached, k is increased by one, and the next further away neighborhood will be searched. [22]

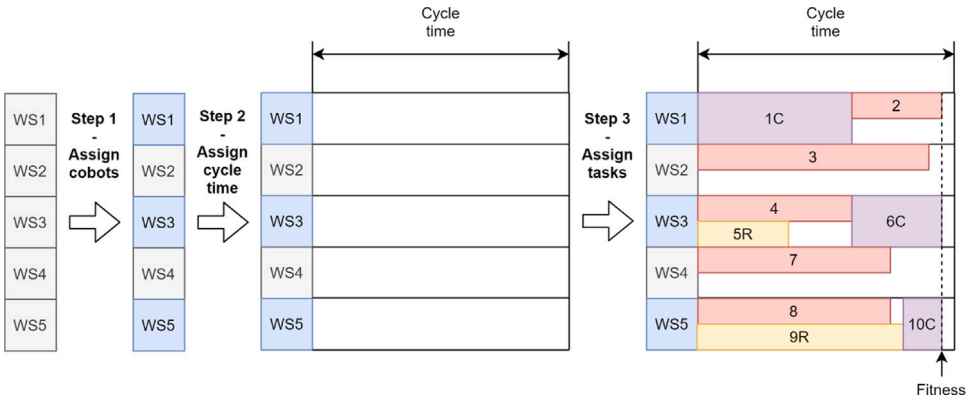

Fig. 8. Biased random-key encoding - Visualize steps.

Algorithm 2
Pseudo code variable neighborhood search.

0	Define k = 1, k_{max}, termination	Initialize the necessary variables
1	while(k <= k_{max})	Terminate once k is larger than k_{max}
2	while(!termination)	Do until a specific number of solutions is evaluated or time limit is reached
3	x' = N_k(x)	Get neighboring solution with k changes
4	if(x' is better than x)	If a better solution has been found
5	x = x'	Set x to new best
6	k = 1	Start vns again from new best with k = 1
7	go to line 1	Start next iteration of the main loop
8	end if	
9	end while	
10	k++	Increase k by one
11	end while	

4.4. Hybrid genetic algorithm

In Algorithm 3, the basic concept of the evaluation method of the hybrid genetic algorithm is shown. The code for the genetic algorithm is similar to the code from Algorithm 1. Whenever the evaluated method from Algorithm 3 is called, in line 1, we see that for every solution s that is passed to this method, a fitness value x is generated. In line 2, it is checked if this fitness is within a certain range of the best fitness found so far. If this is the case, a variable neighborhood search on this solution is initiated. Pretests have shown that the best results can be found if the variable neighborhood search is applied to solutions that are within 10% of the best solution found so far.

To generate neighboring solutions k, changes are applied to the initial encoded solution. Two different variants of changes are used to generate neighboring solutions:

- **Basic change**
  For the biased random-key encoding, one change is one value of the vector that is replaced by a random real value between 0 (inclusive) and 1 (exclusive).
- **Intelligent change**
  The first step of the intelligent change is to determine if the change affects the task to workstation assignment, the task priority, or the cobot to workstation assignment.
  - Task to workstation assignment
    All workstations that are available in the workstation group where the task should be produced are ordered by a factor that is the multiplicative of production speed and production cost. A rank-based selection is used to favor workstations with a low factor.
  - Task priority
    This is the same as for the basic change.
  - Cobot to workstation assignment
    All workstations from the base solution get a rank assigned based on the created costs and a second rank that is based on the production duration. Both rankings have the same chance to be used for the cobot to workstation assignment. Workstations that created high costs and workstations that have long production times are favored for a cobot assignment in the intelligent change.

Pretests have shown that the best results are found with 90% intelligent changes and 10% basic changes. The k values that are used in this algorithm are 1, 3, and 5. In line 5 of Algorithm 3, it can be seen that 50 neighboring solutions are generated for each k value. In the lines from 8 to 12, it can be seen that a first improvement strategy is used. This means that if any generated solution is better than the original solution, this solution becomes the new starting solution and the algorithm is restarted with k = 1. If no improvement can be found in any of the 50 generated solutions of one k value, k is increased by 2. In Algorithm 3 in the lines 18 to 20, it can be seen that the fitness and individual are stored, if they are better than the best individual that has been found so far.

5. Numerical experiments

5.1. Dataset dimensions

The first dataset used for the computational studies is the dataset provided by an industry partner. This dataset is used for the combined cobot assignment and job shop scheduling problem and has the following metrics:

- 54 workstations
- 210 orders
- 1265 tasks

It contains many typical elements of a job shop scheduling problem. The dataset pertains to the production of mechanical parts such as engines, pumps, and housings. The workstations are in the following areas:

- Heat treatment furnaces
- Prefabrication
- Assembly

The current version of the evaluation is assumed to be fully deterministic.

Based on the real-world data set, 50 artificial data sets have been added to compare the CP model with the hybrid genetic algorithm. These artificial data sets have the following metrics:

- 30/50 workstations
- 50/100 orders

Algorithm 3
Pseudo code - hybrid genetic algorithm evaluation.

	Parameters:	Parameters at the start of the program
	BestSolution	Best solution found so far
	BestFitness	Best fitness found so far
	EvaluateSolution()	Method to get the quality of a passed individual (depending on the problem)
	VnsThreshhold	Threshhold to check if the variable neighborhood shearch should be applied
0	Evaluate(solution s)	
1	x = EvaluateSolution(s)	
2	if(x ≤ BestFitness * VnsThreshhold)	
3	k = 1, k_{max} = 5	
4	while(k ≤ k_{max})	
5	for(i = 0, i ≤ 50, i++)	
6	s' = N_k(x)	
7	x' = EvaluateSolution(s')	
8	if(x' < x)	
9	x = x'	
10	s = s'	
11	k = 1	
12	goto line 3	
13	end if	
14	end for	
15	k += 2	
16	end while	
17	end if	
18	if(x ≤ BestFitness)	
19	BestFitness = x	
20	BestSolution = s'	
21	end if	
22	return x	

- 300/600/1200 tasks

The amount of workstations is varied and it depends on the number of different workstations used in the real-world data set. The data sets with 1200 tasks are similar to the real-world data set and additionally, there are two smaller versions with 600 and 300 tasks. The amount of orders has been reduced to 50 and 100 for the task amount of 300/600 and 1200, respectively. This has been done to increase the number of precedence relations. A full description of the generation of these data sets can be found in Appendix A.

The second dataset used for the computational studies is the assembly line balancing problem with cobot assignment used in [1]. The dataset contains the following metrics:

- 3 problem sizes (small: 20 tasks; medium: 50 tasks; large: 100 tasks)
- 50 problems with 10 different parameter settings per problem

The following additional parameters were introduced to create the 10 different parameter settings:

- **Robot flexibility (RF):** The percentage of all tasks that can be done by a robot
- **Collaboration flexibility (CF):** The percentage of all tasks that can be done by the human in collaboration with the robot
- **West ratio (WR):** The average number of tasks per workstation
- **Robot density (RD):** The percentage of workstations that have a cobot assigned

In Appendix B, the 10 different parameter settings for each problem can be seen. To compare the developed algorithm with the existing results in the literature, we used the first 30 instances of each size.

5.2. Overview

The first computational study is used to find a good solution for the real-world problem provided by our industry partner. Therefore, the following variations are compared on the combined cobot assignment and job shop scheduling problem:

Table 6
Variations in the first computational study.

Algorithm	Encoding	Cobots
Genetic algorithm	Integer encoding	0
Genetic algorithm	Biased random-key encoding	0
Genetic algorithm	Integer encoding	5
Genetic algorithm	Biased random-key encoding	5

- No cobots assigned, 5 cobots assigned
- Biased random-key encoding, integer encoding

For each instance of the dataset, this results in four variations. These can be seen in Table 6.

The goal of these calculations is to find out which algorithm/encoding works best on the real-world problem and how much improvement can be made with five cobots. The best algorithm/encoding combination is then used to show how much improvement can be made with the first cobot and how much this decreases with the deployment of an additional cobot, i.e., we evaluate the marginal utility of cobots. After analyzing the real-world problem, the best working algorithm is compared to a CP solver running the model presented in Section 3 on the artificial data set. This comparison demonstrates the strengths and weaknesses of the genetic algorithm in comparison to the CP formulation.

In the final analysis, the developed algorithm is applied to a similar problem (combined cobot assignment and assembly line balancing problem) from the literature. Solving this problem should show that our developed algorithm can compete with state-of-the-art algorithms used on benchmark problems in the literature.

The algorithms/encodings were implemented using the programming language C#. The integer-based encoding is represented by an array of integer values. For each position of the vector, a lower and upper bound is encoded as an integer value. In the biased random-key encoding, a solution representation is a vector of double values with the lower bound being 0 and the upper bound being 1 (excluded). This double represents a real number with 8 bytes of memory. This means it has a precision of 15–17 digits [23].

This simple representation allows metaheuristics such as a genetic algorithms to easily create new solutions. This solution encoding has already been successfully used for several classical optimization problems (including job shop scheduling problems), as well as real-world applications [11].

The framework used to implement the encoding and run the genetic algorithm was HeuristicLab. HeuristicLab is a framework for heuristic and evolutionary algorithms that can be easily extended using a plugin-based architecture [24]. The CP model described in Section 3 was implemented and run using IBM ILOG CP Optimizer 12.10 as a commercial solver. For all calculations, a computer with an Intel i7-8700 3.20 GHz CPU is used.

5.3. Combined cobot assignment and job shop scheduling problem

5.3.1. Real-world data

To increase the number of different test cases, three different versions of the large real-world dataset are calculated:

- Full dataset
- First and second half of the dataset
- Four quarters of the dataset

The full dataset, the halves, and the quarters of the data were evaluated independently, respectively. The data is divided by adding orders until e.g., half of the tasks are in the data set half one. In all the versions, the deployment of five cobots is compared to the deployment of no cobots. Based on [1], it can be assumed that a cobot will increase the production speed of a workstation by 30%, which will, in turn, lead to a cost reduction of 30%. The datasets in Tables 1 and 2 are named based on the following schema:

- "Item set identifier"_"Minimum job"-"Maximum job"

An example for this naming is "I2_1-637," which means that the unique identifier for the dataset is I2 and the jobs 1 to 637 have to be produced in this data set.

This leads to a total of seven different test sets. For each test set, with and without cobots, the biased random-key and the integer encoding was run 10 times.

The value received from the objective function is the fitness value assigned to a specific solution. Makespan and production costs are normalized in a way, that the normalized value is closer to 1 if it is closer to the minimum. If these two values are summed up, the fitness ranges from zero (worst possible solution) to two (minimal cost and makespan).

For the following results, the integer-based encoding has the abbreviation "Int" and the biased random-key encoding has the abbreviation "Real". The hybrid genetic algorithm and the genetic algorithm have the following parameters set:

- Mutation rate: 5%
- Elite Solution: 1
- Individuals per generation: 100
- Maximization of the normalized value

In the first computational study, the integer-based encoding of the genetic algorithm is compared to the biased random-key encoding. To make a fair comparison between the two algorithms, both stop after a set time limit. These time limits change for different instance sizes and can be seen in Table 7.

The lower bounds of one instance of the real-world data set have been calculated by letting the hybrid genetic algorithm minimize both the makespan and the cost separately with five cobots (three runs with a time limit of 300 min). The upper bounds have been calculated by maximizing the makespan and the cost three times (for 300 min each) with no cobots present.

Table 7
Time limits for the real-world data set (in minutes).

Duration	Full	Halves	Quarters
Short	100	30	10
Medium	200	60	20
Long	300	90	30

In Fig. 9, the average solution quality over all runs of the genetic algorithm with zero and five cobots have been reported. The real-encoded version yielded better results over all data sets and is on an average 9.7% better than the integer encoded version. The full data can be found in Appendix C.

Since the biased random-key encoding delivered better results over all data sets, it is used as a base for the hybrid genetic algorithm and is additionally compared to the results of the CP model.

Based on the findings of the first experiment, a hybrid genetic algorithm has been started with the parameters described for the genetic algorithm and with the following additional parameters that are necessary for the variable neighborhood search. Based on experiments, the variable neighborhood search is applied only when a newly generated solution is within 10% of the best solution found so far. To generate neighboring solutions, 90% of the changes are intelligent changes and 10% are basic changes. The distance to the neighborhood solutions started with $k = 1$ change and was increased by 2 if no better solution could be found within the first 50 generated individuals. The variable neighborhood search stopped once k became larger than 5. This means that if a solution is within 10% of the best-found solution, at least 150 individuals in the neighborhood are generated and evaluated.

In Fig. 10, the average solution quality from the hybrid genetic algorithm with five cobots with standard deviation is reported. The time limits from Table 7 were also used for these experiments. By using the hybrid genetic algorithm, the results from the genetic algorithm with biased random-key encoding could be improved by another 2%.

The values in Fig. 10 for the halves and quarters are the average values over both halves and all four quarters, respectively. The full data can be found in Appendix D.

In Fig. 11, the average solution quality of the hybrid genetic algorithm without cobot over ten runs can be seen. Upon comparison of these results with those from Fig. 10, where the genetic algorithm selects the workstations and five cobots should be deployed, it can be seen that the solution quality drastically increases with the usage of these cobots. Without cobots, the average solution quality is at 1.218 while the solution quality with five cobots averages at 1.737. This means an average improvement of 42.6% can be reached in these scenarios. The full data can be found in Appendix D.

To better understand the influence of cobots on the solution quality, one instance was selected to evaluate an increasing number of cobots. Therefore, the first half of the data set I2_1-637 was selected. For each cobot, the instance was computed ten times.

In Fig. 12, the fitness value for different numbers of cobots can be seen. It can be seen that the first deployed cobot has the highest impact on the solution quality. The full data can be found in Appendix E.

In Fig. 13, the percentage improvement resulting from different cobot numbers can be seen. The first cobot increases the fitness by 34.9%. The second cobot still improves the objective function by 15.9%. This decreases to 1.7% for the third, 1.7% for the fourth, and 0.02% for the fifth cobot. These effects correspond to the law of decreasing marginal returns.

The hybrid genetic algorithm with biased random-key encoding performs well on the real-world data set in comparison to the genetic algorithm. To prove its capability to solve the combined cobot assignment and job shop scheduling problem, the hybrid genetic algorithm is compared to the CP solver running the formulation stated in Section 3 on the real-world instance. For all real-world instances, the CP solver is granted the same amount of time as the long run of the hybrid genetic

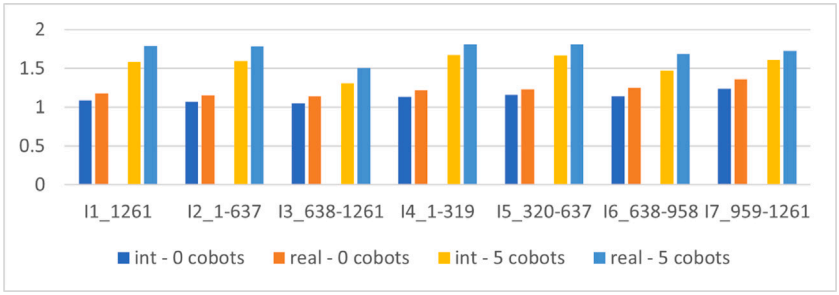

Fig. 9. Average solution quality - GA.

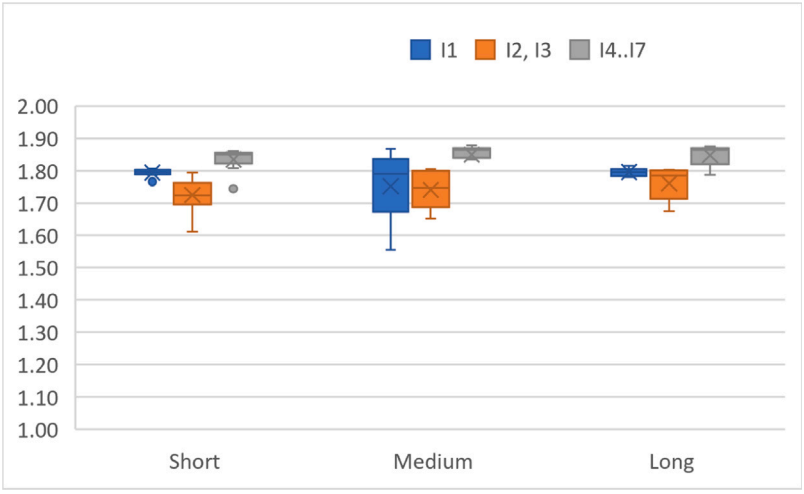

Fig. 10. Average results with standard deviation - Hybrid genetic algorithm - 5 Cobots.

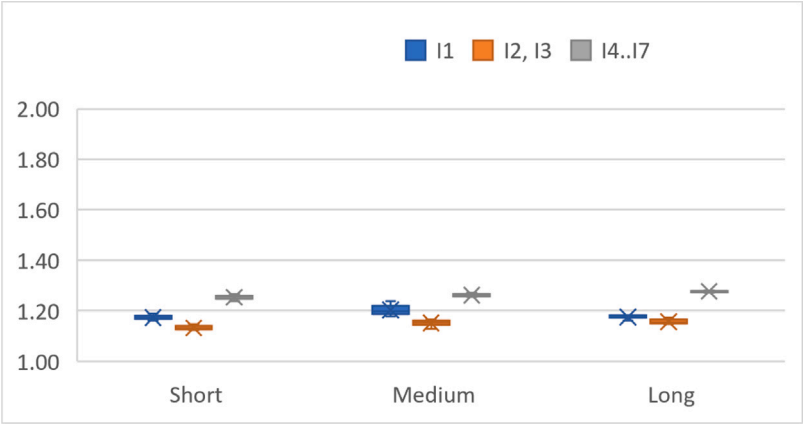

Fig. 11. Average results with standard deviation - Hybrid genetic algorithm - 0 Cobots.

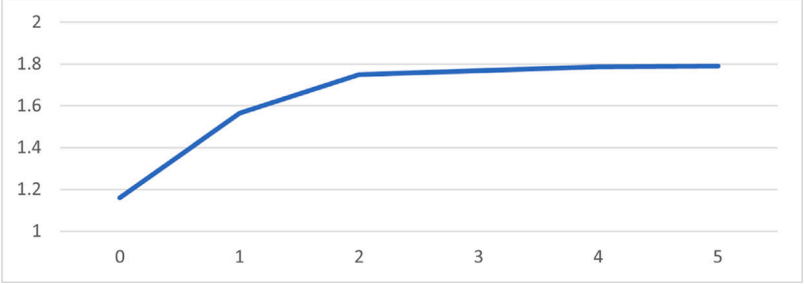

Fig. 12. I2_1-637 - Normalized fitness per cobot.

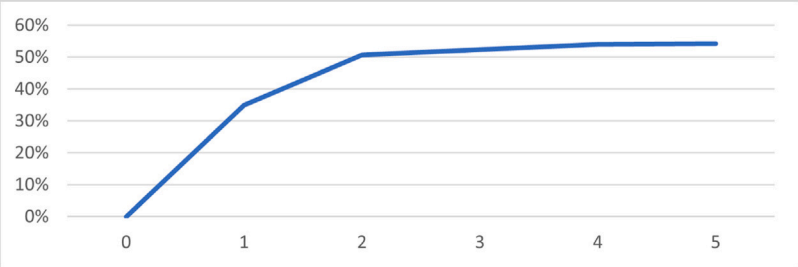

Fig. 13. I2_1-637 - Percentage improvement per cobot.

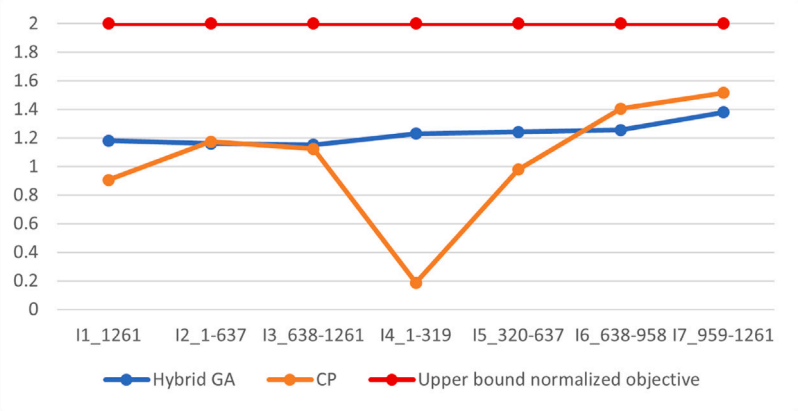

Fig. 14. Comparison CP/Hybrid GA - 0 cobots.

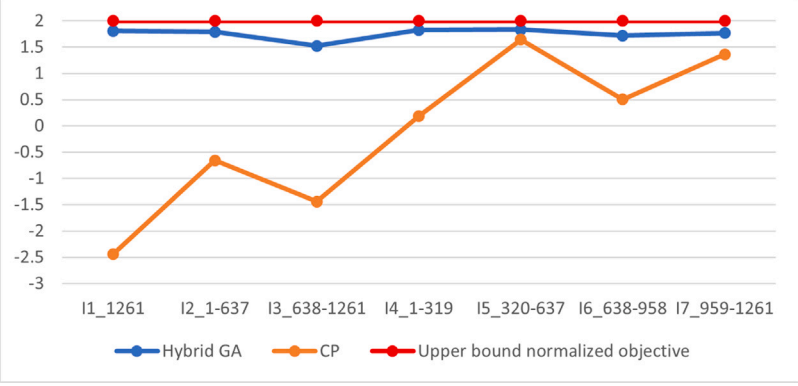

Fig. 15. Comparison CP/Hybrid GA - 5 cobots.

algorithm from Table 7 (second column). If the CP solver calculates the fitness for zero cobots of a specific data set, it is compared to the average solution quality of the hybrid genetic algorithm of this specific data set with zero cobots.

In Fig. 14, the hybrid genetic algorithm is compared to the CP formulation with 0 cobots to assign. In this basic version, the CP solver is able to find better solutions than the hybrid genetic algorithm in three of seven instances. When looking at all data sets, the CP model performs on average 15.6% worse than the hybrid genetic algorithm. If the outlier in data set I4 is neglected, the solution quality is only decreased by 4% compared to the average solution quality reported by the hybrid genetic algorithm.

In Fig. 15, the solution quality of the hybrid genetic algorithm is compared to the solution quality obtained from the CP model with five cobots to assign. Due to the increased complexity of the cobot to workstation assignment in the model, the CP solver is not able to find solutions that can compete with the solutions from the hybrid genetic algorithm. It is also worth noting that the best results found with the CP model are for the four quarters I4 to I7 and hence the smallest data sets, whereas the worst result is achieved for the full data set.

To allow for an even more thorough assessment of the solution quality achieved by the hybrid genetic algorithm, we conducted additional CP solver runs involving considerably increased computational resources and time limits. The idea was to specifically account for the cobot-assisted scenario, in which the CP solver fails to deliver objective function values greater than zero for the full and the two halved data sets. The results of these experiments, as shown in Fig. 20 of Appendix G, indicate that the CP solution quality could be notably improved but is still not sufficient to beat the hybrid GA in the cobot-assisted scenario. Only for the 0 cobots case, the CP solutions are consistently better than those provided by the hybrid GA.

5.3.2. *Artificial data sets*

To prove that the hybrid genetic algorithm is able to solve the combined cobot assignment and job shop scheduling problem, an additional 50 artificial data sets in five categories have been created.

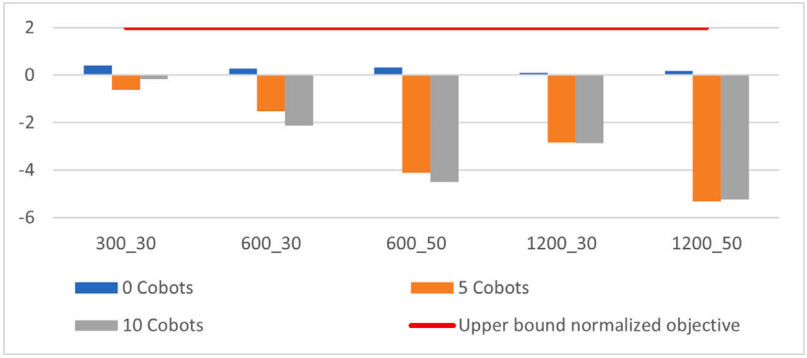

Fig. 16. Comparison - Artificial instances - CP model.

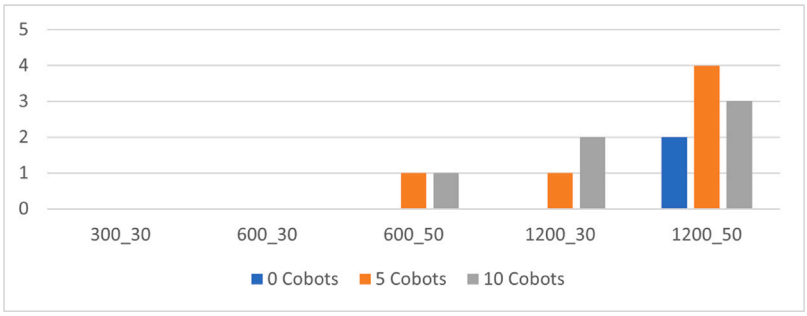

Fig. 17. Comparison - Artificial instances - No CP solution found (number of occurrences).

The data sets are named based on the amount of included tasks and workstations according to the scheme "Tasks"_"Workstations". These sets are grouped into three sizes:

- Small: 300_30
- Medium: 600_30, 600_50
- Large: 1200_30, 1200_50

The run time for the small, medium, and large data set is 60, 180, and 300 min, respectively. The data sets are explained in detail in Appendix A. If the CP model calculates the fitness for zero cobots of a specific data set, it is compared to the average solution quality of the hybrid genetic algorithm of this specific data set with zero cobots.

For each artificial data set, the bounds for the normalized objective function were computed in the following way: Upper bounds on makespan and cost were taken from short runs of the CP solver, stopping immediately as soon as the first feasible solution was found. The lower bound on the makespan was derived from a parallel machine relaxation of the problem, using the mixed-integer programming formulation of [25] as a basis. The lower bound on the cost was again retrieved from the CP solver, in the form of the initial lower bound on the cost objective.

In Fig. 16, the average solution quality achieved by the CP model is shown. What can be seen here is that due to the complexity increase when allowing the CP model to place cobots, it is unable to find better solutions. The main factors are the amount of tasks and the amount of workstations

In Fig. 17, it can be seen that the increased model complexity due to an increasing amount of tasks and workstations will decrease the ability of the CP solver to find solutions. It can be seen that the 600_30 instance is way easier to solve than the 600_50 instance, since the average solution quality is way better and the CP solver is unable to find a solution in two cases for the 600_50 instance. This increases to 9 instances for the 1200_50 instance. In Fig. 18, the average solution quality of the hybrid genetic algorithm is shown. It can be observed that the hybrid genetic algorithm generates good results for all cases and is able to effectively utilize the additional cobots to improve the solution quality. The average solution quality for the individual runs of the CP model and the hybrid genetic algorithm with zero, five, and ten cobots can be found in Appendix F.

Similar to the real-world instances, we also performed a comparison between the hybrid GA's results and those obtained from extended CP runs. The details of these experiments and the associated comparison can be found in Appendix G (Fig. 21). Despite the markedly increased computational effort, the CP solutions are only slightly better than those delivered by the hybrid GA with the original, restricted time limits.

5.4. Combined cobot assignment and assembly line balancing problem

The second computational analysis should compare the developed algorithm with the existing methods in the literature. Therefore, the algorithm is compared to the genetic algorithm and the mixed integer programming developed for [1]. Most of the small instances have been solved optimally by the mixed integer programming. Therefore, the goal is to reach the optimal solution or to come as close to it as possible. This is different for the large instances, as the complexity increases, it is not possible anymore to solve them optimally within a reasonable amount of time. For the large instances, the results have to be compared to those from the genetic algorithm developed in [1].

As stated above, the first 30 instances of each size are used for the comparison. In the literature dataset, the genetic algorithm was run until no improvement could be reached within 1000 generations. For these runs, an average run time over all instances has been reported.

For easy instances of one size, the average run time will be too long and the algorithm might find the best solution very fast. However, for hard instances, the algorithm might have less time than the genetic algorithm used in [1]. The average run time used in the literature is as follows:

- **Small instances:** 114 s
- **Medium instances:** 666 s
- **Large instances:** 2994 s

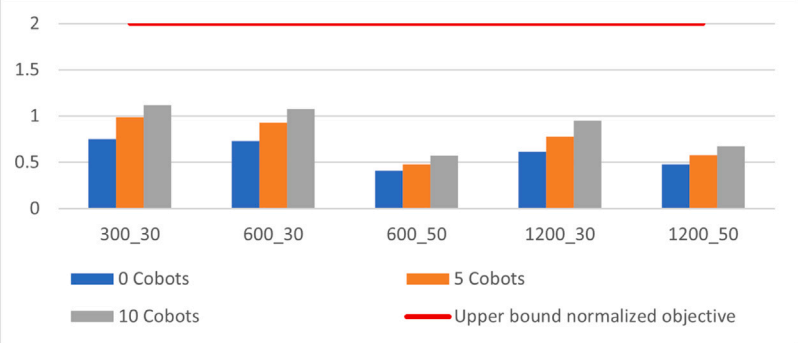

Fig. 18. Comparison - Artificial instances - hybrid genetic algorithm.

The mixed integer programming has the following run time limitations:

- **Small instances:** 7200 s
- **Medium instances:** 7200 s
- **Large instances:** 28800 s

For our computational experiments, we use a similar, however, fixed time limitation. This should make our results easier to reproduce and better comparable for other researchers' study purposes. Based on the results from the first computational experiment, we know that the additional cobots will greatly increase the complexity of the problem. The used time limits were as follows:

- **Small instances:** 150 s
- **Medium instances:** 1000 s
- **Large instances:** 3600 s

The following results were created using the hybrid genetic algorithm with the biased random-key encoding on the literature dataset. In the literature dataset, each instance was calculated 10 times and the best result was reported. Therefore, in this computational study, the same rules apply–each calculation is carried out 10 times and the best result is reported. For the following results, the hybrid genetic algorithm is the algorithm that was developed in this paper. The genetic algorithm is the algorithm that was used in [1].

The performance value that is used to compare the hybrid genetic algorithm with the genetic algorithm or the mixed integer programming from the literature is calculated by dividing the hybrid genetic algorithms solution through the solution of the genetic algorithm or mixed integer programming. A comparison, for example is that all small instances are generated by averaging this value over all calculated instances.

In the small dataset, we can assume that in all cases where we found the result from the mixed integer programming, we identified the best possible result. Upon comparison with the genetic algorithm, it was discovered that the reported genetic algorithm results could be improved 4 times, our hybrid genetic algorithm tied with the genetic algorithm 22 times, and 4 times, no solution as good as the reported genetic algorithm solution could be found.

When we look at the quality over all the 30 instances, the hybrid genetic algorithm can be found within 0.2% of the genetic algorithm and within 0.7% of the mixed integer programming.

The robot density is the main factor that increases computational complexity for this dataset. With a robot density of 0, no cobot can be assigned to workstations. This means that these instances should be easier to solve than those with a high robot density, where the algorithm has to decide where to place cobots and how tasks are produced on workstations with cobots.

In Table 8, the results of the hybrid genetic algorithm are grouped by the three different robot densities that have been used in this study. With the average computational time, the less complex instances with a robot density of 0 could be easily found and even improved. For complex instances with a robot density of 0.4, just the average time is not enough; therefore, the results are worse than those found in the literature. All computed data values can be found in Appendix H.

Table 8
Comparison to the small dataset based on robot density.

Robot density	0	0.2	0.4
GA	99.73%	100.08%	100.56%
MIP	100.00%	100.57%	101.30%

Table 9
Comparison to the medium dataset based on robot density.

Robot density	0	0.2	0.4
GA	99.93%	100.78%	101.93%
MIP	100.20%	100.44%	101.53%

Table 10
Comparison to the large dataset based on robot density.

Robot density	0	0.2	0.4
GA	98.88%	100.87%	102.65%
MIP	100.75%	98.92%	103.28%

In the medium dataset, only three instances could be improved. However, even if the best results are not found as often as in the literature dataset, the algorithm is only 1% worse than the results of the genetic algorithm and 0.8% worse than the results of mixed integer programming.

In Table 9, the results are grouped again by robot density. Similar to the results from the small dataset, the algorithm can find better or comparable results for instances with a robot density of 0 or 0.2 but does not find as good results for a robot density of 0.4. The full data can be found in Appendix I.

In the seven cases, the reported results from the large instances of the genetic algorithm could be improved. The overall solution quality is 1.2% worse than the solution quality of the genetic algorithm and 1.3% worse than the solution quality of the mixed integer programming. Please note that the mixed integer programming only found solutions for 17 out of the 30 instances.

In Table 10, the results of the large dataset are grouped by robot density. The results are quite similar to the results of the small and medium datasets. For instances with a robot density of 0 and 0.2, the algorithm delivers even better results than the genetic algorithm and the mixed integer programming from the literature. For instances with a robot density of 0.4, the run time is not long enough to find comparable results. The full data can be found in Appendix J.

To allow a fairer comparison of our hybrid genetic algorithm with the algorithm from the literature on instances with a robot density of 0.4, all such instances are calculated again with doubled time limits (small: 300 s; medium: 2000s; large: 7200s). The reported quality develops as follows: Small instances:

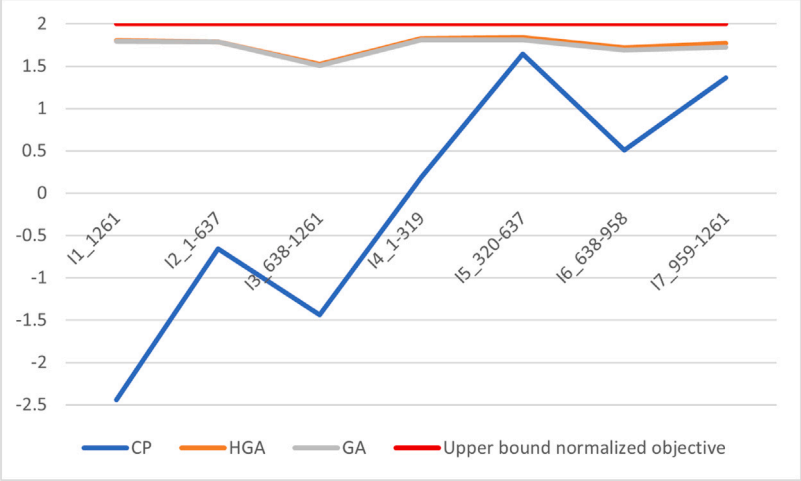

Fig. 19. Comparison GA/HGA/CP - 5 cobots.

- **GA:** 100.56% ⇒ 99.53%
- **MIP:** 101.30% ⇒ 100.24%

Medium instances:

- **GA:** 101.93% ⇒ 101.38%
- **MIP:** 101.53% ⇒ 100.99%

Large instances:

- **GA:** 102.65% ⇒ 102.02%
- **MIP:** 103.28% ⇒ 102.78%

It can be seen that with a fairer time limit, the results come quite close to those reported in the literature dataset. This means even if our hybrid genetic algorithm was developed for a real-world dataset, it can compete with state-of-the-art algorithms from the literature. The full computational results can be found in Appendices K–M.

6. Main findings

In this paper, a new combined cobot assignment and job shop scheduling problem was introduced that should be solved based on data from a real-world company. The optimization goal was to minimize the makespan and the production costs simultaneously. For this purpose, we employed a combined objective function based on a normalization scheme.

In the first computational experiments, where a genetic algorithm was used to solve this problem, an integer based encoding was compared to a biased random-key encoding. In these experiments, the results from the biased random-key encoding were 9.7% better than the results from the integer encoding.

Based on these findings, a hybrid genetic algorithm with biased random-key encoding was proposed. This hybrid genetic algorithm combines the exploratory strengths of a genetic algorithm with the exploitatory strengths of a variable neighborhood search. To further improve the strengths of the variable neighborhood search, changes to the current solution are based on properties of the base solution. This means, when generating a neighboring solution, it is more likely that a cobot is assigned to a workstation that had high costs or a long makespan in the original solution.

In Fig. 19, it can be seen that the genetic algorithm and the hybrid genetic algorithm manage to produce good solutions for the real-world problem with 5 cobots. However, with the improvements, the hybrid genetic algorithm is able to improve the solutions of the genetic algorithm by another 2%. The implemented CP model is able to solve the problem to a certain degree. Especially small instances without cobots to assign can be solved in a way that they can compete with the solutions from the hybrid genetic algorithm.

An additional important finding of this paper is that the first deployed cobot in a production environment has the highest impact on the objective function. In the investigated real-world data set, the objective function could be improved by 35% by the first cobot. This value quickly decreases, with additional cobots that are deployed to the production environment.

To have an additional comparison between the hybrid genetic algorithm and the CP formulation, 50 artificial data sets have been created. The results on these data sets look similar to the results from the real-world instances. Small instances without cobots to assign can be solved pretty well with the CP model. However, this changes drastically when the problem setting allows more cobots to be assigned.

The best working algorithm from the first computational experiment, the hybrid genetic algorithm with biased random-key encoding, is changed in a way that it is able to solve a similar problem, namely the cobot assignment and assembly line balancing problem from the literature. A major problem in this comparison is that the complexity of the instances fluctuates greatly. The focus of our comparison was on the complex instances with cobots to be assigned and therefore we admitted longer but fixed computation times. With these considerations, the developed hybrid genetic algorithm is able to compete with the results reported in the literature.

7. Outlook

The real-world problem and data set used is a representative data set for medium- to large-sized job shop scheduling problems. It should be possible to achieve similar results with other real-world data sets.

When applied to other real-world problems, the objective function might not be clear. Therefore, hybrid multi-objective algorithms such as the NSGA-II in [26] could be used to optimize multiple objective values and generate a Pareto optimal front (solutions that are not dominated by other solutions) that could be presented to a domain expert.

The developed hybrid genetic algorithm with the biased random-key encoding delivered very good results for both the combined cobot assignment and job shop scheduling and the combined cobot and assembly line balancing problems. Additionally, environmental uncertainties can be included in further research. It might be interesting to see how the flexibility of cobots can be used to assist bottleneck workstations in case of machine breakdown.

CRediT authorship contribution statement

Alexander Kinast: Conceptualization, Methodology, Software, Data curation, Writing – original draft. **Roland Braune:** Methodology, Software, Formal analysis, Writing – review & editing. **Karl F. Doerner:** Supervision, Conceptualization, Writing – review & editing. **Stefanie Rinderle-Ma:** Supervision, Conceptualization, Writing – review & editing. **Christian Weckenborg:** Resources, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We are thankful that the RISC Software GmbH allowed us to use the simulation framework, Easy4Sim. This simulation framework was used to assign a fitness value to an individual in the genetic algorithm. Additionally, we are grateful for the real-world dataset provided in cooperation with our industry partner.

Appendix A. Artificial data sets

To have additional data sets for the comparison between the hybrid genetic algorithm and the CP model, 50 artificial data sets in five categories have been created. In the following Table, these categories can be seen. In the paper, the categories will be named "Tasks"_ "Workstations". An example would be 300_30 which is the first category in the table. For each data set, solutions with zero, five, and ten cobots have been created. This means that the hybrid genetic algorithm, as well as the CP model have evaluated 150 solutions for this comparison. The run time is 60, 180, and 300 min for data sets with 300, 600, and 1200 tasks, respectively.

Workstations	Workstation groups	Orders	Tasks
30	10	50	300
30	10	50	600
50	10	50	600
30	10	100	1200
50	10	100	1200

The instances are created in such a way that they are not easy to solve. In comparison to the real-world instances (where workstations might be bottlenecks depending on the current orders), all workstation groups have an equal amount of tasks assigned. Tasks are generated with the following properties:

- Amount: 1–10
- Production time: 100–200
- 50% chance for setup time in the range: 30–100
- 50% chance for de-setup time in the range: 30–100

Workstations are generated with the following properties:

- Production type
  - 50%: Serial workstation with capacity 1
  - 50%: Parallel workstation with capacity 2–6
- Time factor: 0.75–1.25
- Cost factor: 0.75–1.25

If more than one part is produced successively on one capacity of a workstation, the setup time is only necessary before the first amount that is produced and the de-setup time is only necessary after the last produced amount.

Appendix B. Parameter settings literature

In the following table, the different settings for the literature data set can be seen. The west ratio defines the number of workstations, based on the instance size. An example would be a west ratio (average number of tasks per workstation) of 2 in a small instance with 20 tasks. This would lead to 10 workstations. The robot density (percentage of workstations that have a cobot) will define how much workstation can get a cobot assigned. The robot flexibility (percentage of tasks that can be done by a cobot) and the collaborative flexibility (percentage of tasks that can be done collaborative) define the share of tasks that can be done by the robot or in collaboration.

Scenario	RF	CF	West ratio	Robot density
1	0	0	2	0
2	0.2	0.2	2	0.2
3	0.4	0.4	2	0.2
4	0.2	0.2	2	0.4
5	0.4	0.4	2	0.4
6	0	0	4	0
7	0.2	0.2	4	0.2
8	0.4	0.4	4	0.2
9	0.2	0.2	4	0.4
10	0.4	0.4	4	0.4

Appendix C. Computational results of the genetic algorithm on the real-world data set

The following Table C.1 shows the average solution quality of the genetic algorithm (integer and real encoding) over ten runs for zero and five cobots.

Appendix D. Computational results of the hybrid genetic algorithm on the real-world data set

The following Table D.1 shows the average solution quality of the hybrid genetic algorithm over ten runs for zero and five cobots.

Appendix E. Solution quality per number of cobots

In the following table, the fitness values for the data set I2_1-637 with changing numbers of cobots can be seen.

Cobots	0	1	2	3	4	5
Run 1	1.15	1.57	1.76	1.77	1.78	1.79
Run 2	1.17	1.58	1.77	1.77	1.79	1.79
Run 3	1.16	1.56	1.78	1.75	1.80	1.77
Run 4	1.17	1.57	1.76	1.75	1.79	1.80
Run 5	1.16	1.56	1.76	1.79	1.79	1.78
Run 6	1.15	1.57	1.77	1.77	1.79	1.79
Run 7	1.16	1.57	1.77	1.78	1.79	1.79
Run 8	1.16	1.57	1.77	1.77	1.77	1.79
Run 9	1.15	1.53	1.77	1.76	1.78	1.79
Run 10	1.16	1.57	1.58	1.77	1.79	1.80
Average	1.16	1.56	1.75	1.77	1.79	1.79

Appendix F. Artificial instances - GA/CP

In the following table, the average objective value for all instances of one category of the artificial data set can be seen.

Datasets	300_30	600_30	600_50	1200_30	1200_50
GA - 0 cobots	0.75	0.73	0.41	0.61	0.48
GA - 5 cobots	0.98	0.93	0.48	0.78	0.58
GA - 10 cobots	1.12	1.08	0.57	0.95	0.67
CP - 0 cobots	0.41	0.28	0.32	0.10	0.19
CP - 5 cobots	−0.61	−1.51	−4.11	−2.85	−5.31
CP - 10 cobots	−0.17	−2.14	−4.49	−2.87	−5.24

Table C.1

Data set	I1_1-1261	I2_1-637	I3_638-1261	I4_1-319	I5_320-637	I6_638-958	I7_959-1261
int - 0 cobots	1.09	1.07	1.05	1.14	1.16	1.14	1.24
real - 0 cobots	1.18	1.15	1.14	1.22	1.23	1.25	1.36
int - 5 cobots	1.58	1.60	1.31	1.68	1.67	1.48	1.61
real - 5 cobots	1.79	1.78	1.50	1.81	1.81	1.69	1.73

Table D.1

Hybrid GA	I1_1261	I2_1–637	I3_638–1261	I4_1–319	I5_320–637	I6_638–958	I7_959-1261
0 Cobots	1.18	1.15	1.14	1.22	1.23	1.25	1.36
5 Cobots	1.80	1.79	1.50	1.81	1.82	1.70	1.74

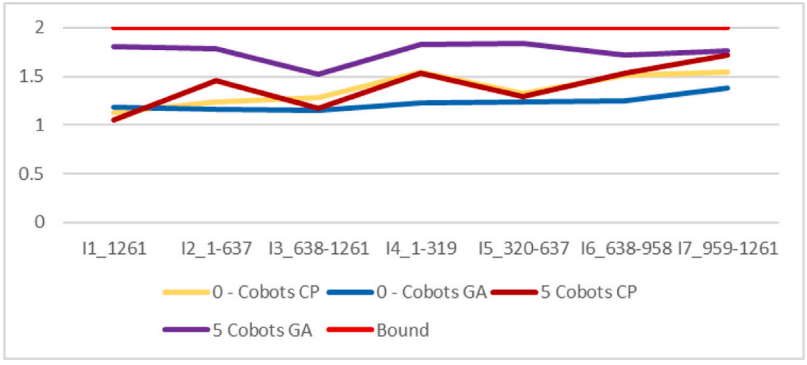

Fig. 20. Hybrid GA vs. CP (long run): normalized objective function values for real instances.

Appendix G. Hybrid GA vs. CP long runs - a comparison based on normalized objective function values

In addition to the CP runs using the same time limits as for the hybrid GA, we also ran the CP solver with strongly boosted computational resources and relaxed time limits. Preliminary attempts showed that it was not possible to achieve substantial improvements when relying on the original normalized objective function. Therefore, we adopted the following two-stage approach: for each instance without cobots, we first performed a pure makespan minimization run for twelve hours using six parallel worker threads. In a second step, the CP model is run with the objective of cost minimization but with an upper bound constraint on the makespan. The upper bound is set to the best makespan found during the first stage. Note that this process can be considered a *lexicographic* approach as known from multi-objective optimization, however, with the single-objective problems not solved to optimality. The reason for the relative order of the two objectives is that when minimizing the makespan, the production costs stay within reasonable bounds. When minimizing just the costs (without any constraint on the makespan), it becomes immediately clear from the model definition that only the assignment of tasks to workstations and machines matters, without any consideration of start and completion times. Hence, the resulting schedule will not be usable.

To accelerate the second stage CP runs, we used the solutions from stage one to "warm-start" the solver. Furthermore, we fed the first stage solutions obtained for the zero-cobots scenario into the runs based on five and ten cobots (again as warm-start solutions) to make sure that the solver always finds at least one feasible solution.

Based on the stage two results, the individual objectives are finally combined to a normalized objective value in the same fashion as described in Sections 5.3.1 and 5.3.2 for the real-world and the artificial data set, respectively. Figs. 20 and 21 give an overview of the obtained average normalized values. Note that the results reported for the hybrid GA were obtained using the original restricted time limits and thus coincide with those presented in Figs. 14, 15 and 18.

For Appendices J–M, the computational results from the small instances can be seen. The first six columns are used to identify the data set from the literature. The last three columns show the best result that have been found by:

- GA: Genetic algorithm from the literature
  - GA Gap: Gap between the genetic algorithm and the best found solution
- MIP: Mixed integer programming from the literature
  - MIP Gap: Gap between the mixed integer programming and the best found solution
- HGA: Hybrid genetic algorithm that has been developed in this paper
  - HGA Gap: Gap between the hybrid genetic algorithm and the best found solution

Appendix H. Computational results small data set

See Table H.1.

Appendix I. Computational results medium data set

See Table I.1.

Appendix J. Computational results large data set

See Table J.1.

Appendix K. Robot density 0.4 - small instances

See Table K.1.

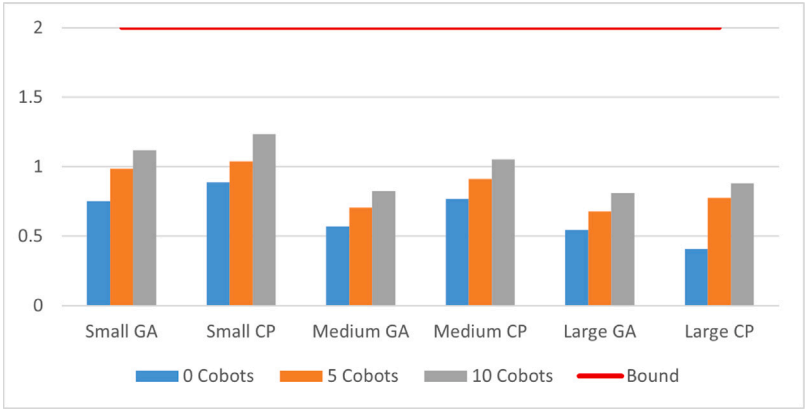

Fig. 21. Hybrid GA vs. CP (long run) - normalized objective function values for artificial instances.

Table H.1

Id	Stations	Size	Robots	RF	CF	GA	MIP	HGA	GA Gap	MIP Gap	HGA Gap
442	5	20	0	20	20	595	595	595	0	0	0
442	5	20	1	20	20	587	587	587	0	0	0
442	5	20	2	20	20	536	536	536	0	0	0
442	10	20	0	20	20	343	343	343	0	0	0
442	10	20	2	20	20	322	320	320	2	0	0
442	10	20	4	20	20	320	320	320	0	0	0
442	5	20	1	40	40	568	568	568	0	0	0
442	5	20	2	40	40	522	522	522	0	0	0
442	10	20	2	40	40	309	309	309	0	0	0
442	10	20	4	40	40	305	291	299	14	0	8
441	5	20	0	20	20	580	580	580	0	0	0
441	5	20	1	20	20	556	556	556	0	0	0
441	5	20	2	20	20	506	506	506	0	0	0
441	10	20	0	20	20	321	321	321	0	0	0
441	10	20	2	20	20	321	321	321	0	0	0
441	10	20	4	20	20	321	321	321	0	0	0
441	5	20	1	40	40	556	556	556	0	0	0
441	5	20	2	40	40	506	506	506	0	0	0
441	10	20	2	40	40	321	321	321	0	0	0
441	10	20	4	40	40	321	321	321	0	0	0
165	5	20	0	20	20	576	576	576	0	0	0
165	5	20	1	20	20	528	526	528	2	0	2
165	5	20	2	20	20	489	489	489	0	0	0
165	10	20	0	20	20	307	302	302	5	0	0
165	10	20	2	20	20	298	285	293	13	0	8
165	10	20	4	20	20	262	262	281	0	0	19
165	5	20	1	40	40	526	524	530	2	0	6
165	5	20	2	40	40	489	488	489	1	0	1
165	10	20	2	40	40	277	277	284	0	0	7
165	10	20	4	40	40	270	260	274	10	0	14

Table I.1

Id	Stations	Size	Robots	RF	CF	GA	MIP	HGA	GA Gap	MIP Gap	HGA Gap
455	13	50	0	20	20	457	456	456	1	0	0
455	13	50	3	20	20	424	422	426	2	0	4
455	13	50	5	20	20	416	416	423	0	0	7
455	25	50	0	20	20	304	304	304	0	0	0
455	25	50	5	20	20	304	304	304	0	0	0
455	25	50	10	20	20	304	304	304	0	0	0
455	13	50	3	40	40	417	417	422	0	0	5
455	13	50	5	40	40	398	398	409	0	0	11
455	25	50	5	40	40	304	304	304	0	0	0
455	25	50	10	40	40	304	304	304	0	0	0
454	13	50	0	20	20	568	568	568	0	0	0
454	13	50	3	20	20	522	540	527	0	18	5
454	13	50	5	20	20	506	506	525	0	0	19
454	25	50	0	20	20	396	396	396	0	0	0
454	25	50	5	20	20	396	396	396	0	0	0

(*continued on next page*)

Table I.1 (*continued*).

Id	Stations	Size	Robots	RF	CF	GA	MIP	HGA	GA Gap	MIP Gap	HGA Gap
454	25	50	10	20	20	396	396	396	0	0	0
454	13	50	3	40	40	520	522	527	0	2	7
454	13	50	5	40	40	496	504	513	0	8	17
454	25	50	5	40	40	293	293	300	0	0	7
454	25	50	10	40	40	293	293	299	0	0	6
53	13	50	0	20	20	937	924	935	13	0	11
53	13	50	3	20	20	855	858	873	0	3	18
53	13	50	5	20	20	818	825	858	0	7	40
53	25	50	0	20	20	560	560	560	0	0	0
53	25	50	5	20	20	560	560	560	0	0	0
53	25	50	10	20	20	560	560	560	0	0	0
53	13	50	3	40	40	854	862	862	0	8	8
53	13	50	5	40	40	811	829	848	0	18	37
53	25	50	5	40	40	560	560	560	0	0	0
53	25	50	10	40	40	560	560	560	0	0	0

Table J.1

Id	Stations	Size	Robots	RF	CF	GA	MIP	HGA	GA Gap	MIP Gap	HGA Gap
451	25	100	0	20	20	1046	1024	1036	22	0	12
451	25	100	5	20	20	991		1003	0		12
451	25	100	10	20	20	946	972	998	0	26	52
451	50	100	0	20	20	605	566	566	39	0	0
451	50	100	10	20	20	568		566	2		0
451	50	100	20	20	20	612		566	46		0
451	25	100	5	40	40	962	1057	995	0	95	33
451	25	100	10	40	40	908	944	972	0	36	64
451	50	100	10	40	40	595		566	29		0
451	50	100	20	40	40	570		566	4		0
328	25	100	0	20	20	576	565	575	11	0	10
328	25	100	5	20	20	525	537	545	0	12	20
328	25	100	10	20	20	501	504	530	0	3	29
328	50	100	0	20	20	322	322	322	0	0	0
328	50	100	10	20	20	322		322	0		0
328	50	100	20	20	20	322		322	0		0
328	25	100	5	40	40	526	540	541	0	14	15
328	25	100	10	40	40	486	500	520	0	14	34
328	50	100	10	40	40	322		322	0		0
328	50	100	20	40	40	322		322	0		0
19	25	100	0	20	20	912	906	920	6	0	14
19	25	100	5	20	20	848	879	878	0	31	30
19	25	100	10	20	20	813	826	857	0	13	44
19	50	100	0	20	20	548	548	548	0	0	0
19	50	100	10	20	20	548		548	0		0
19	50	100	20	20	20	548		548	0		0
19	25	100	10	40	40	791	820	838	0	29	47
19	25	100	10	40	40	791	820	838	0	29	47
19	50	100	10	40	40	548		548	0		0
19	50	100	20	40	40	548		548	0		0

Table K.1

Id	Stations	Size	Robots	RF	CF	GA	MIP	HGA	GA Gap	MIP Gap	HGA Gap
442	5	20	2	20	20	536	536	536	0	0	0
442	10	20	4	20	20	320	320	320	0	0	0
442	5	20	2	40	40	522	522	522	0	0	0
442	10	20	4	40	40	305	291	299	14	0	8
441	5	20	2	20	20	506	506	506	0	0	0
441	10	20	4	20	20	321	321	321	0	0	0
441	5	20	2	40	40	506	506	506	0	0	0
441	10	20	4	40	40	321	321	321	0	0	0
165	5	20	2	20	20	489	489	489	0	0	0
165	10	20	4	20	20	262	262	281	0	0	19
165	5	20	2	40	40	489	488	489	1	0	1
165	10	20	4	40	40	270	260	274	10	0	14

Appendix L. Robot density 0.4 - medium instances

See Table L.1.

Appendix M. Robot density 0.4 - large instances

See Table M.1.

Table L.1

Id	Stations	Size	Robots	RF	CF	GA	MIP	HGA	GA Gap	MIP Gap	HGA Gap
455	13	50	5	20	20	416	416	423	0	0	7
455	25	50	10	20	20	304	304	304	0	0	0
455	13	50	5	40	40	398	398	409	0	0	11
455	25	50	10	40	40	304	304	304	0	0	0
454	13	50	5	20	20	506	506	525	0	0	19
454	25	50	10	20	20	396	396	396	0	0	0
454	13	50	5	40	40	496	504	513	0	8	17
454	25	50	10	40	40	293	293	299	0	0	6
53	13	50	5	20	20	818	825	858	0	7	40
53	25	50	10	20	20	560	560	560	0	0	0
53	13	50	5	40	40	811	829	848	0	18	37
53	25	50	10	40	40	560	560	560	0	0	0

Table M.1

Id	Stations	Size	Robots	RF	CF	GA	MIP	HGA	GA Gap	MIP Gap	HGA Gap
451	25	100	10	20	20	946	972	998	0	26	52
451	50	100	20	20	20	612		566	46		0
451	25	100	10	40	40	908	944	972	0	36	64
451	50	100	20	40	40	570		566	4		0
328	25	100	10	20	20	501	504	530	0	3	29
328	50	100	20	20	20	322		322	0		0
328	25	100	10	40	40	486	500	520	0	14	34
328	50	100	20	40	40	322		322	0		0
19	25	100	10	20	20	813	826	857	0	13	44
19	50	100	20	20	20	548		548	0		0
19	25	100	10	40	40	791	820	838	0	29	47
19	50	100	20	40	40	548		548	0		0

References

[1] C. Weckenborg, K. Kieckhäfer, C. Müller, M. Grunewald, T.S. Spengler, Balancing of assembly lines with collaborative robots, Bus. Res. 13 (1) (2020) 93–132, http://dx.doi.org/10.1007/s40685-019-0101-y.

[2] K. Wegener, W.H. Chen, F. Dietrich, K. Dröder, S. Kara, Robot assisted disassembly for the recycling of electric vehicle batteries, in: The 22nd CIRP Conference on Life Cycle Engineering, Vol. 29, 2015, pp. 716–721, http://dx.doi.org/10.1016/j.procir.2015.02.051.

[3] A. Weigl-Seitz, K. Hohm, M. Seitz, H. Tolle, On strategies and solutions for automated disassembly of electronic devices, Int. J. Adv. Manuf. Technol. 30 (5–6) (2006) 561–573, http://dx.doi.org/10.1007/s00170-005-0043-8.

[4] C. Banu, B. Serol, A research survey: review of AI solution strategies of job shop scheduling problem, J. Intell. Manuf. 26 (2013) 961–973, http://dx.doi.org/10.1007/s10845-013-0837-8.

[5] M.K. Amjad, S.I. Butt, R. Kousar, R. Ahmad, M.H. Agha, Z. Faping, N. Anjum, U. Asgher, Recent research trends in genetic algorithm based flexible job shop scheduling problems, Math. Probl. Eng. 2018 (2018) http://dx.doi.org/10.1155/2018/9270802.

[6] G. Zhang, L. Zhang, X. Song, Y. Wang, C. Zhou, A variable neighborhood search based genetic algorithm for flexible job shop scheduling problem, Cluster Comput. 22 (5) (2019) 11561–11572, http://dx.doi.org/10.1007/s10586-017-1420-4.

[7] Q. Djamila, S. Petrovic, A survey of dynamic scheduling in manufacturing systems, J. Sched. 12 (2009) 417–431, http://dx.doi.org/10.1007/s10951-008-0090-8.

[8] F. Carole, P. Sanja, A genetic algorithm for the real-world fuzzy job shop scheduling, Lecture Notes in Comput. Sci. (2005) http://dx.doi.org/10.1007/11504894_71.

[9] B. Cunha, A.M. Madureira, B. Fonseca, D. Coelho, Deep reinforcement learning as a job shop scheduling solver: A literature review, 2020, http://dx.doi.org/10.1007/978-3-030-14347-3_34, HIS 2018, AISC 923, 350–359.

[10] A. Kinast, K.F. Doerner, S. Rinderle-Ma, Biased random-key genetic algorithm for cobot assignment in an assembly/disassembly job shop scheduling problem, Procedia Comput. Sci. 180 (2021) 328–337, http://dx.doi.org/10.1016/j.procs.2021.01.170.

[11] M.L. Lucena, C.E. Andrade, M.G. C.Resende, F.K. Miyazawa, Some extensions of biased random-key genetic algorithms, in: Proceedings of the XLVI Symposium of the Brazilian Operational Research Society, Salvador, Brazil, 2014, http://www.din.uem.br/sbpo/sbpo2014/pdf/arq0357.pdf. (Accessed 26 November 2021).

[12] N. Sridhar, M.V. Raj, K.C. Sekar, Minimizing manufacturing cost in flexible job-shop scheduling problems, in: International Journal of Artificial Intelligence and Mechatronics, Vol. 1, 2013, pp. 2320–5121, http://www.ijaim.org/download/conference/ICEA/Mech-1_Final.pdf. (Accessed 26 November 2021).

[13] B. Qu, P. Suganthan, Multi-objective evolutionary algorithms based on the summation of normalized objectives and diversified selection, Inform. Sci. 180 (17) (2010) 3170–3181, http://dx.doi.org/10.1016/j.ins.2010.05.013, Including Special Section on Virtual Agent and Organization Modeling: Theory and Applications.

[14] A. Otto, C. Otto, A. Scholl, Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing, European J. Oper. Res. 228 (1) (2013) 33–45, http://dx.doi.org/10.1016/j.ejor.2012.12.029.

[15] Y.N. Sotskov, N.V. Shakhlevich, NP-hardness of shop-scheduling problems with three jobs, Discrete Appl. Math. 59 (3) (1995) 237–266, http://dx.doi.org/10.1016/0166-218X(95)80004-N.

[16] B. Chen, C. N.Potts, J.W. Gerhard, A Review of Machine Scheduling: Complexity, Algorithms and Approximability, Kluwer Academic Publishers, 1998, pp. 21–169, http://dx.doi.org/10.1007/978-1-4613-0303-9_25.

[17] C. Blum, J. Puchinger, G. Raidl, A. Roli, et al., A brief survey on hybrid metaheuristics, in: Proceedings of BIOMA, 2010, pp. 3–18, https://www.researchgate.net/publication/228365733_A_brief_survey_on_hybrid_metaheuristics. (Accessed 26 November 2021).

[18] S. Meeran, M. Morshed, A hybrid genetic tabu search algorithm for solving job shop scheduling problems: A case study, J. Intell. Manuf. 23 (4) (2012) 1063–1078, http://dx.doi.org/10.1007/s10845-011-0520-x.

[19] L. Gao, G. Zhang, L. Zhang, X. Li, An efficient memetic algorithm for solving the job shop scheduling problem, Comput. Ind. Eng. 60 (4) (2011) 699–705, http://dx.doi.org/10.1016/j.cie.2011.01.003.

[20] M. Affenzeller, S. Wagner, S. Winkler, A. Beham, Simulating evolution: Basics about genetic algorithms, in: Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications, CRC Press, 2009, pp. 0–10, http://dx.doi.org/10.1201/9781420011326.

[21] A. Scholl, Balancing and Sequencing of Assembly Lines, Physica-Verlag HD, 1999.

[22] N. Mladenović, P. Hansen, Variable neighborhood search, Comput. Oper. Res. 24 (11) (1997) 1097–1100, http://dx.doi.org/10.1016/S0305-0548(97)00031-2.

[23] Microsoft documentation, 2021, https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/floating-point-numeric-types. (Accessed 26 November 2021).

[24] S. Wagner, G. Kronberger, A. Beham, M. Kommenda, A. Scheibenpflug, E. Pitzer, S. Vonolfen, M. Kofler, S. Winkler, V. Dorfer, M. Affenzeller, Architecture and design of the HeuristicLab optimization environment, in: Advanced Methods and Applications in Computational Intelligence, Springer, 2014, pp. 197–261, http://dx.doi.org/10.1007/978-3-319-01436-4_10.

[25] S. Martello, F. Soumis, P. Toth, Exact and approximation algorithms for makespan minimization on unrelated parallel machines, Discrete Appl. Math. 75 (2) (1997) 169–188, http://dx.doi.org/10.1016/S0166-218X(96)00087-X.

[26] M. Frutos, A.C. Olivera, F. Tohmé, A memetic algorithm based on a NSGAII scheme for the flexible job-shop scheduling problem, Ann. Oper. Res. 181 (1) (2010) 745–765, http://dx.doi.org/10.1007/s10479-010-0751-9.