universität wien

DISSERTATION / DOCTORAL THESIS

Titel der Dissertation / Title of the Doctoral Thesis

„Combining metaheuristics and process mining to improve manufacturing processes“

verfasst von / submitted by

Alexander Kinast, BSc. MSc.

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Doktor der Wirtschaftswissenschaften (Dr.rer.oec.)

Wien, 2023 / Vienna, 2023

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 796 305 175
Dissertationsgebiet lt. Studienblatt / field of study as it appears on the student record sheet:	Wirtschaftsinformatik
Betreut von / Supervisor:	Univ.-Prof. Mag. Dr. Karl Franz Dörner, Privatdoz.
Betreut von / Supervisor:	Prof. Dr. Stefanie Rinderle-Ma

Acknowledgements

I am thankful to my two supervisors Karl F. Dörner and Stefanie Rinderle-Ma. Over the years, in countless meetings, both gave me valuable scientific feedback and great advice. Karl's passion for optimization and logistic topics, as well as Stefanie's enthusiasm for process mining, forged the foundation of this work.

A special thanks belong to Roland Braune. He assisted with my PhD wherever possible. Especially in the programming and computational area, whenever I had new results, he helped to interpret them, asked critical questions, and helped me improve my problem-solving skills.

I am incredibly thankful to my family for their unconditional support. My parents, for furthering my education and supporting me in all life situations. My brother and his wife, for interesting discussion and correction reading.

However, the biggest thank belongs to my partner Katharina who was with me during all years of this work and even before. She supported me in all life situations and also helped me during this work with countless correction readings. Additionally, she spent hours understanding my work and helping everywhere possible.
I am proud that you will be with me as my wife!

Lastly, I want to thank all the others who contributed directly and indirectly to this work, especially my friends.

Abstract

Industry and logistics currently experience a significant shift towards novel technologies such as cooperative robots (cobots). These cobots can share a workspace with a human worker without needing a safe zone and cooperate with humans in different activities. Additionally, these cobots have short setup times which lead to more flexible production than ever before.
Existing scheduling problems do not consider these flexible resources. Therefore, the first aim of this thesis is to extend existing scheduling problems with a cobot to workstation assignment which can be done before a planning period starts.
The second aim of this work is to create an algorithm that can solve the newly introduced problems. Therefore, after introducing this extended cobot assignment and scheduling problems, existing metaheuristics, like a genetic algorithm, can be used to generate solutions. To rate the quality of the results generated by the metaheuristic, the problems are solved with a constraint programming formulation, and the results are compared.
The rapid development described before affects not only the production environments in the industry but also algorithms used in different practical applications and research areas. Over the last few years process mining has found a large number of applications. Process mining describes a large set of algorithms which create a process model from a process event log.
This work's final aim is to combine metaheuristics and process mining algorithms. It is shown that a memetic algorithms performance can be boosted through a novel combination with a process mining algorithm.

Zusammenfassung

Industrie und Logistik erleben derzeit einen deutlichen Wandel hin zu neuartigen Technologien wie kooperativen Robotern. Diese kooperativen Roboter können sich einen Arbeitsbereich mit einem menschlichen Arbeiter teilen und mit diesem in verschiedenen Aktivitäten zusammenarbeiten, ohne dass ein Sicherheitsabstand benötigt wird. Durch kurze Setupzeiten führt dies zu einer flexibleren Produktion als je zuvor.
Bestehende Scheduling-Probleme berücksichtigen diese flexiblen Ressourcen nicht. Daher ist das erste Ziel dieser Arbeit, bestehende Scheduling-Probleme so zu erweitern, dass vor einem neuen Planugszeitraum kooperative Roboter auf die Arbeitsstationen zugewiesen werden.
Das zweite Ziel dieser Arbeit ist es, einen Algorithmus zu erstellen, der die neu erstellten Probleme lösen kann. Um die neu erstellten Scheduling-Probleme mit Zuweisung von kooperativen Robotern zu Arbeitsstation zu lösen, können bestehende Metaheuristiken wie ein genetischer Algorithmus eingesetzt werden. Um die Qualität der Ergebnisse des genetischen Algorithmus zu bewerten, wird eine Constraint-Programmierung Formulierung durchgeführt und die Ergebnisse werden verglichen.
Die zuvor beschriebene rasante Entwicklung betrifft nicht nur die Produktionsumgebungen in der Industrie, sondern auch Algorithmen, die in verschiedenen praktischen Anwendungen und Forschungsgebieten verwendet werden. In den letzten Jahren wurde eine große Anzahl an Anwendungen für Process-Mining gefunden. Process-Mining beschreibt eine große Menge an Algorithmen, welche ein Prozessmodell aus einem Prozessprotokoll erstellen.
Das finale Ziel der Arbeit ist es, Metaheuristiken mit Process-Mining-Algorithmen zu kombinieren. Es wird hier gezeigt, dass die Leistung eines memetischen Algorithmus durch eine neuartige Kombination mit Process-Mining-Algorithmen gesteigert werden kann.

Contents

Acknowledgement I

Abstract II

Zusammenfassung III

1 Motivation 1
- 1.1 Introduction . . . 1
- 1.2 Collaborative robots (cobots) . . . 1
- 1.3 Extended cobot assignment and scheduling problems . . . 2
  - 1.3.1 Metaheuristics . . . 3
  - 1.3.2 Process mining . . . 3
- 1.4 Research aims . . . 5
- 1.5 Research questions . . . 5
- 1.6 List of contributions . . . 7
  - 1.6.1 Contribution A . . . 8
  - 1.6.2 Contribution B . . . 8
  - 1.6.3 Contribution C . . . 9
- 1.7 Thesis structure . . . 9

2 Methodology and state of research 11
- 2.1 Scheduling problems . . . 11
- 2.2 Metaheuristics . . . 12
- 2.3 Process mining . . . 13
- 2.4 Metaheuristics and process mining . . . 13
- 2.5 Process mining as visualization tool . . . 14

3 Biased random-key genetic algorithm for cobot assignment in an assembly/disassembly job shop scheduling problem 15
- Abstract . . . 15
- 3.1 Introduction . . . 17
- 3.2 Problem description . . . 17
- 3.3 Related work and research contribution . . . 19
- 3.4 Solution method . . . 19
- 3.5 Real world data set . . . 21
- 3.6 Preliminary results . . . 22
- 3.7 Conclusion and outlook . . . 24
- 3.8 Acknowledgements . . . 25
- References . . . 25

4 A hybrid metaheuristic solution approach for the cobot assignment and job shop scheduling problem 26
- Abstract . . . 27
- 4.1 Introduction . . . 27

4.1.1 Overview . . . 27
4.1.2 Related work and research contribution . . . 28
4.2 Problem description . . . 29
4.2.1 Cobot assignment and job shop scheduling . . . 29
4.2.2 Cobot assignment and assembly line balancing . . . 30
4.3 A constraint programming formulation of the job shop scheduling problem with cobot assignment . . . 30
4.4 Solution method . . . 32
4.4.1 Overview . . . 32
4.4.2 Genetic algorithm . . . 32
4.4.3 Variable neighborhood search . . . 34
4.4.4 Hybrid genetic algorithm . . . 35
4.5 Numerical experiments . . . 35
4.5.1 Dataset dimensions . . . 35
4.5.2 Overview . . . 36
4.5.3 Combined cobot assignment and job shop scheduling problem . . . 37
4.5.4 Combined cobot assignment and assembly line balancing problem . . . 40
4.6 Main findings . . . 42
4.7 Outlook . . . 42
CRediT authorship contribution statement . . . 43
Acknowledgements . . . 43
Appendix A. Artificial data sets . . . 43
Appendix B. Parameter settings literature . . . 43
Appendix C. Computational results of the genetic algorithm on the real-world data set . . . 43
Appendix D. Computational results of the hybrid genetic algorithm on the real-world data set . . . 43
Appendix E. Solution quality per number of cobots . . . 43
Appendix F. Artificial instances - GA/CP . . . 43
Appendix G. Hybrid GA vs. CP long runs - a comparison based on normalized objective function values . . . 44
Appendix H. Computational results small data set . . . 44
Appendix I. Computational results medium data set . . . 44
Appendix J. Computational results large data set . . . 44
Appendix K. Robot density 0.4 - small instances . . . 44
Appendix L. Robot density 0.4 - medium instances . . . 46
Appendix M. Robot density 0.4 - large instances . . . 46
References . . . 47

5 Combining metaheuristics and process mining: Improving cobot placement in a combined cobot assignment and job shop scheduling problem 48
Abstract . . . 49
5.1 Introduction . . . 50

5.1.1 Problem description and Solution technique 50
5.2 Solution method 51
5.2.1 Process Mining 51
5.2.2 Design Choices 52
5.2.3 Implementation 53
5.3 Preliminary results 53
5.3.1 Attribute visualization 53
5.3.2 Scenario 1 - exclusive choice 54
5.3.3 Scenario 2 - exclusive choice and sequence 54
5.3.4 Scenario 3 - workstation groups 55
5.3.5 Scenario 4 - loop 55
5.3.6 Findings 56
5.4 Conclusion and outlook 57
References 57

6 Optimizing the Solution Quality of Metaheuristics through Process Mining based on Selected Problems from Operations Research 59
Abstract 59
6.1 Introduction 59
6.2 Related Work 62
6.3 Fundamentals 63
6.3.1 Memetic Algorithms (MA) 63
6.3.2 Local process model mining 63
6.4 Solution Method 64
6.4.1 Operation Research Problems 64
6.4.2 Encoding and evaluation 65
6.4.3 Local process model mining 66
6.4.4 Memetic algorithm 67
6.5 Numerical experiments 69
6.5.1 Data and code 69
6.5.2 Constraint programming formulation 69
6.5.3 Data dimensions 70
6.5.4 Real-world cobot assignment and job shop scheduling problem 70
6.5.5 Generated cobot assignment and job shop scheduling problem 71
6.5.6 Cobot assignment and flexible job shop scheduling problem 72
6.6 Summary and Outlook 74
Acknowledgments 74
References 74

7 Conclusion 77
7.1 Results 77
7.2 Discussion 78
7.3 Outlook 79

List of Figures

1 Solution for a job shop scheduling problem 2
2 Aim of the thesis 5
3 Contributions of the thesis 7
3.1 Human-cobot interaction 17
3.2 Precedence graph 18
3.4 Biased random-key encoding 20
3.4 Decoding and evaluation 21
3.6 Computational results with standard deviation 23
3.6 Aggregated results 23
4.1 Human-cobot interaction 28
4.2 Simple assembly line balancing problem overview 30
4.4 Overview hybrid genetic algorithm 32
4.4 Biased random-key encoding 33
4.4 Decoding and evaluation 33
4.4 Biased random-key encoding - Literature data set 34
4.4 Biased random-key encoding - Visualize steps 35
4.5 Average solution quality - GA 38
4.5 Average results with standard deviation - Hybrid genetic algorithm - 5 Cobots 38
4.5 Average results with standard deviation - Hybrid genetic algorithm - 0 Cobots 38
4.5 I2_1-637 - Normalized fitness per cobot 38
4.5 I2_1-637 - Percentage improvement per cobot 39
4.5 Comparison CP/Hybrid GA - 0 cobots 39
4.5 Comparison CP/Hybrid GA - 5 cobots 39
4.5 Comparison - Artificial instances - CP model 40
4.5 Comparison - Artificial instances - No CP solution found (number of occurrences) 40
4.5 Comparison - Artificial instances - hybrid genetic algorithm . . . 41
4.5 Comparison GA/HGA/CP - 5 cobots 42
4.7 Hybrid GA vs. CP (long run): normalized objective function values for real instances 44
4.7 Hybrid GA vs. CP (long run) - normalized objective function values for artificial instances 45
5.2 Overall approach 52
5.2 Event log file structure 52
5.3 Scenario 1 (Cost: color, Production time: width of the nodes) . . 54
5.3 Scenario 2 (Cost: color, Production time: width of the nodes) . . 55
5.3 Scenario 3 (Cost: color, Production time: width of the nodes) . . 55
5.3 Scenario 3 - 1 cobot (Cost = color, Production time = width of the nodes) 55
5.3 Scenario 4.1 (Cost: color, Production time: width of the nodes) . 56
5.3 Scenario 4.2 (Cost: color, Production time: width of the nodes) . 56

6.1 MA with process discovery knowledge 62
6.4 Integer-based encoding 66
6.4 Generation and usage of local process models 66

List of Tables

3.6 Computational results, with cobots assigned (I1 = full data, I2-I3 = two halves, I4 ... I7 = four quarters) 23
3.6 Computational results, real encoding, five cobots vs. no cobots assigned (I1 = full data, I2-I3 = two halves, I4 ... I7 = four quarters) 24
4.1 Literature comparison 29
4.2 Indices used in the CP formulation 30
4.2 Parameters used in the CP formulation 30
4.3 Interval variables used in the CP model formulation 31
4.3 Sequence variable definition, one for each machine k of workstation ω 31
4.5 Variations in the first computational study 36
4.5 Time limits for the real-world data set (in minutes) 37
4.5 Comparison to the small dataset based on robot density 41
4.5 Comparison to the medium dataset based on robot density . . . 41
4.5 Comparison to the large dataset based on robot density 41
6.5 Detailed results for the real-world problem 71
6.5 Detailed results for the artificially generated instances 72
6.5 Detailed makespan results for the cobot assignment and FJSP . 73

List of Algorithms

4.4 Pseudo code genetic algorithm 32
4.4 Pseudo code variable neighborhood search 35
4.5 Pseudo code - hybrid genetic algorithm evaluation. 36
6.4 Pseudo code - MA with process discovery 68

1 Motivation

1.1 Introduction

In [1], it is described that the traditional industry is experiencing a large shift towards Industry 4.0. Novel technologies are available due to developments in information and communication areas. Connecting many devices and processing a large amount of real-time data is now possible.
In [2], it is stated that technology that arises from this foundation is allowing the industry to grow exponentially. Therefore, these technologies are called disruptive technologies, as they disrupt constant linear growth. These technologies include the Internet of Things, big data, 3D printing, cloud computing, and robots.
In [3], it is explained that the disruptive technologies are the main drivers for Industry 4.0: *(1) the astonishing rise in data volumes, computational power, and connectivity; (2) the emergence of analytics and business-intelligence capabilities; (3) new human–machine interaction; and (4) the transferring of digital instructions to the physical world.*

1.2 Collaborative robots (cobots)

This work focuses on the possibilities and challenges that come with new human-machine interactions in the form of cobots.
Robots, especially cobots, have improved significantly over the last decades. These cobots can share a workspace with a human workforce without needing a safety zone. While sharing the workspace with the human, these cobots can be viewed as colleagues who can do tasks independently or in collaboration with the worker. Additionally, these cobots are designed in a way that they can be easily reprogrammed, even by workers without a programming background. [4]
A significant number of research papers regarding cobots focus on the safety and ergonomics of workers when interacting with cobots. Examples are [5], [6], and [7]. Ensuring and improving the ergonomics and safety of workers is important. However, it might not be the only reason companies buy cobots for production environments.
Improving the productivity of a given production environment might be another major reason for companies to invest in cobots. To quantify the impact of a set number of cobots on existing scheduling problems, these problems must be extended with a cobot-to-workstation assignment. Only a minor amount of research has been done on extended cobot assignment and scheduling problems.

1.3 Extended cobot assignment and scheduling problems

An example of an extended cobot assignment and scheduling problem is [8], where an assembly line balancing problem is extended with a cobot to workstation assignment. When a cobot is assigned to a workstation, the scheduling algorithm can decide if the workload should be done by the human, by the cobot, or in collaboration. A large computational study demonstrates in this paper quantifies the impact of different numbers of cobots.
Another example of a scheduling problem that has been extended with cobots is shown in [9]. In this paper, cobots are added to a flow shop scheduling problem. This problem aims to keep a continuous flow of jobs over a given set of machines. A cobot is then used to assist at least one production machine.
A problem that has not yet been extended with a cobot to workstation assignment is the job shop scheduling problem (JSSP). This work extends a JSSP with a cobot assignment problem.
A simple JSSP problem formulation states jobs, tasks, and machines. Each job consists of a set of tasks that have to be processed on one specific machine. A task can start once the preceding task has finished and the specified machine is free. To complete the task, the workstation is allocated for the whole processing time specified by the task. In this problem, the time required to complete all jobs should be minimized. This time is called makespan. While the problem formulation is pretty simple, this is one of the most challenging np-hard problems, and a 10 x 10 (jobs x machines) instance has been unsolved for 20 years. [10]
Figure 1 shows an example solution for a 3 x 3 problem. Each of the jobs

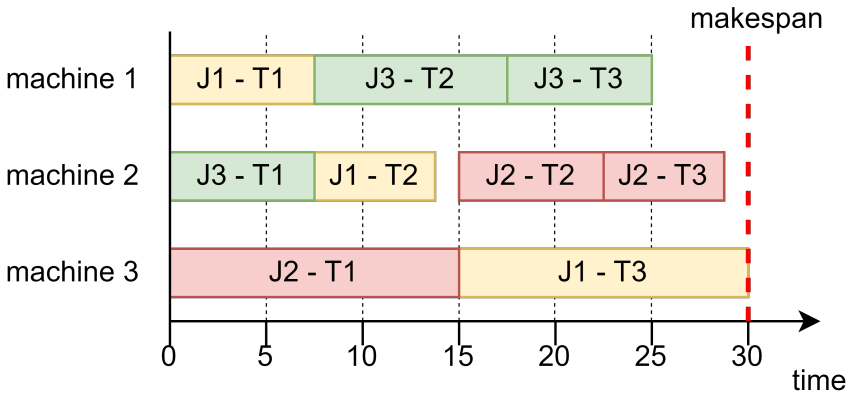


Figure 1: Solution for a job shop scheduling problem

consists of three tasks. Machine two has a break from time units 14 to 15, as task one of job two has not yet finished. It can be seen that the objective value, the makespan (time to complete all jobs) of this solution, is 30.

In [8], it is assumed that a cobot speeds up a workstation by 30%. It might be clear where to place a cobot in a simple problem like the one solved in Figure 1. However, once the objective function and the problem formulation get more complex, the decision of where to place cobots is not clear anymore. It extends the base problem with another decision variable that makes the already hard-to-solve base problem even harder to solve.

1.3.1 Metaheuristics

The complexity of real-world problems is increasing, so it is often impossible to solve them with traditional mathematical methods. Obtaining optimal or nearly-optimal solutions is a challenging task. Metaheuristic algorithms are a set of algorithms that are inspired by nature. These algorithms have been applied successfully to problems of many different domains and have drawn the attention of many researchers. [11]
Well-known examples of these metaheuristics are genetic algorithms [12], evolution strategies [13], and simulated annealing [14].
Genetic algorithms are based on natural selection and reproduction. The algorithm starts with an initial population of individuals (chromosomes). The first step is that fitness is assigned to all individuals. Based on a selection method (a high fitness often improves the chance of being selected), parents for the next generation are selected. A crossover method creates children out of these parents, and a fraction of these children is changed by a mutation method. These children replace the current generation. The idea is that good parent solutions can produce an even better child solution. This means that the average fitness improves over multiple generations, and the algorithm stops once a stopping criterium is reached. [15]
The population aspect of the algorithm allows the exploration of different areas of the search space. [16]
A memetic algorithm (MA) can use this exploration ability of the genetic algorithm and combine it with the exploitation ability of a local search method. In the exploitation, a local search is performed on promising solutions to check if these solutions can be further improved. MAs have been successfully applied to many complex real-world problems and are one example of state-of-the-art metaheuristics. [17]

1.3.2 Process mining

Another research area that gained importance over the last years and has profited a lot from the developments in Industry 4.0 is process mining. Process mining will play an important role in managing business processes and boosting their performance. [18]
Process mining techniques are a set of techniques that allow us to extract knowledge from event log files. These event log files are common in Industry 4.0 processes. The goal of process mining is that real processes should

be discovered, monitored, and improved. Process mining, therefore, combines data mining, business process modelling, and analysis. [19]
Process mining is typically divided into three major categories. The first category is process discovery, where a process model is mined from an event log file. The second category is conformance checking, checking if a log complies with a given process model. In the third category, existing real-world processes should be improved based on given event log files. As process mining techniques focus on the process view, they are way better suited for bottleneck analysis or compliance checking when compared to traditional data mining methods. [20]
Process mining aims to improve operational processes through the systematic use of event data. The key to process mining is the combination of event data and process models. Through this combination, process mining techniques can analyze already happened event logs to find the source of a bottleneck in production or predict the remaining time a process needs to finish. *These can be found in all organizations and industries, including production, logistics, finance, sales, procurement, education, consulting, healthcare, maintenance, and government.* [21] In [22], an example application of process mining is the internal transaction fraud mitigation described. In this application, a process is discovered out of the internal transaction logs of a company. This mined process model can be compared with a given process model to analyze flaws and non-compliant transactions.
In [23], another application of process mining, the analysis of production planning in a global manufacturing company, is demonstrated. Therefore, event logs are extracted from an ERP database, and a process model is discovered and analyzed. The study shows how changes in the production plan can lead to a mismatch between available materials and production needs. With this insight, the production company can better decide between flexibility and efficiency.

1.4 Research aims

Currently, a large number of companies invest in industrial robots. Over the last years, the annual growth was around 11%. This will drastically transform traditional production.[24]
Many of these robots are cobots which are way more flexible than traditional resources. These cobots can be reallocated to bottleneck workstations in production if this bottleneck is known before the next planning period starts. However, most existing scheduling problems do not consider flexible resources, as traditional resources like workstations and new workers need long setup times and training times, respectively. With this background knowledge, we can look at this work's aim.

In Figure 2, an overview of the research aims of this thesis is given. The

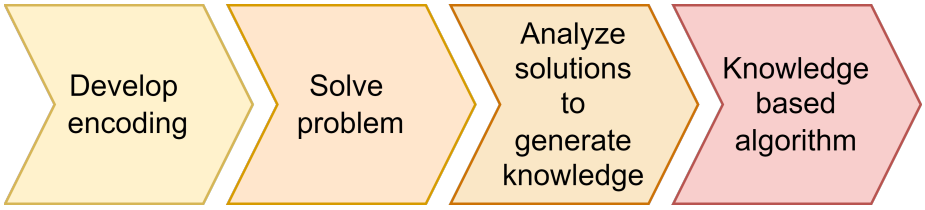


Figure 2: Aim of the thesis

thesis first focuses on extending existing scheduling problems with a cobot-to-workstation assignment. In the second step, state-of-the-art algorithms are adjusted to solve the extended cobot assignment and scheduling problem. Solving different data sets (real-world and literature) gives insight into how much improvement can be expected from a certain amount of cobots in production. Process mining methods are applied to the best solutions in the third step. These methods generate knowledge, like bottleneck workstations and critical workstations. The final aim of this work is to demonstrate the combination of process mining and metaheuristics and show its potential for complex optimization problems. Knowledge extracted with process mining methods is used to improve the results of a state-of-the-art metaheuristic on different data sets of the combined cobot assignment and scheduling problem.
This thesis focuses on the combination of process mining and metaheuristics for combined cobot assignment and scheduling problems. Process mining and metaheuristics have been applied in a large number of applications. Therefore, this combined method should be transferable to other problems. This generalizability is demonstrated at the end of the work.

1.5 Research questions

It is unclear how flexible resources such as cobots can be utilized in scheduling problems. Therefore, existing problem encodings need to be adjusted in a

way that allows the utilization of flexible resources. There are nearly unlimited possibilities for how this encoding/decoding can work. This leads to the first major research question (RQ) with sub-questions:

A) What is the best way to plan resources such as cooperative robots in deterministic manufacturing processes?

A1) How can cobots be assigned to workstations to maximize the objective function?

A2) How much improvement can be reached with a certain amount of cobots?

A3) What is the best number of cobots for a specific system?

Cobots are flexible and can assist nearly every workstation in production. If a company buys new cobots, the first cobot can be placed on the workstation where the objective function is most improved. This means a second cobot already has diminishing returns and adds less value to an existing production environment than the first cobot. Therefore, the goal is to determine the best amount of cobots for a given production system and how much improvement can be reached with a given set of cobots.
When an algorithm optimizes a cobot assignment and scheduling problem, an event log file of specific solutions can be created. This event log file is the basis for many different process mining algorithms. This leads to the second major research question:

B) Can process mining methods generate additional insights into manufacturing processes?

B1) Can the cobot location be predicted based on process mining results (e.g. bottleneck workstations)?

B2) How to visualize metaheuristic solutions with process mining algorithms?

Visualizing metaheuristic results might help to understand the results and learn from them. The question is now if and how good process mining can be used to visualize the solutions generated by a metaheuristic. Additionally, the question arises if the results generated by these process mining algorithms can be used to predict bottlenecks and allow intelligent placement of cobots.
The final research question is:

C) Can process mining algorithms improve the performance of metaheuristics?

C1) Can the knowledge extraction be automated during the run of the metaheuristic?

C2) What is the best way to incorporate knowledge in the metaheuristic?

Since process mining methods are good at finding bottlenecks and critical workstations, mined information might benefit metaheuristics. The question is now, can this information improve metaheuristics, and what is the best way to do so? If it is beneficial for the solution quality, it might be a good idea to run the process mining on different solutions over the run of the metaheuristic. Does this overhead pay off?

1.6 List of contributions

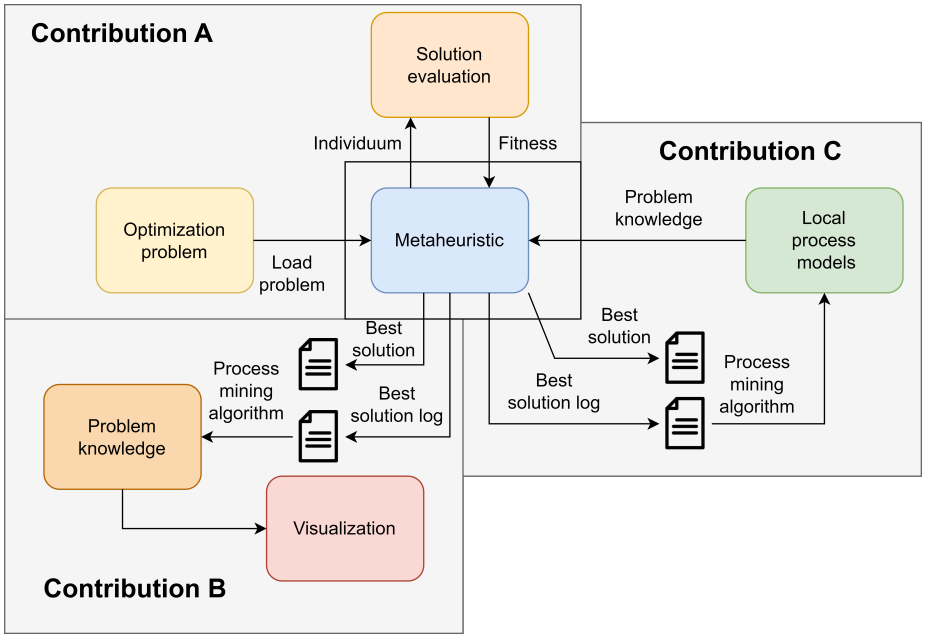


Figure 3: Contributions of the thesis

Figure 3 gives an overview of the main contribution done in this thesis. The first main contribution, **Contribution A**, seen at the left top of Figure 3, is how an algorithm can load and optimize an optimization problem extended with a cobot to workstation assignment. At the left bottom of Figure 3, the second main contribution of this work, **Contribution B**, can be seen. This contribution shows how process mining algorithms can generate knowledge about solutions generated by a genetic algorithm. It is also explored how these process mining methods can visualize generated knowledge. The third main contribution is visualised on the right side of Figure 3, **Contribution C**. It shows process mining algorithms and knowledge generated with these algorithms can interact with a memetic algorithm to boost its performance.

1.6.1 Contribution A

It is not clear how existing scheduling problems can be extended with a cobot to workstation assignment. In this part of the work, the extended cobot assignment and job shop scheduling problem is introduced. Two encodings are developed, which allow the optimization of this problem with metaheuristics like a genetic algorithm. Additionally, a constraint programming formulation is done for the problem. Through a numerical study, the performance of the genetic algorithm and the impact of a certain number of cobots on a given production environment is analyzed. This is in detail described in Chapter 3 and Chapter 4. Both chapters try to answer the research questions **RQ A, A1, A2, A3** and the contribution is the following:

1. This thesis introduces the combined cobot assignment and job shop scheduling problem and solves the combined cobot assignment and assembly line balancing problem from the literature. (sections 3.2 and 4.2)

2. Two encodings are developed, a biased random key and an integer encoding that can be used to optimize the problem. These can be used to encode and decode one solution of a combined cobot assignment and scheduling problem.(sections 3.4.2 and 4.4.2)

3. Developed a genetic algorithm to solve the combined cobot assignment and job shop scheduling problem (section 3.4). Developed a hybrid genetic algorithm to solve both problems (section 4.4).

4. In this thesis, a constraint programming formulation is created for the combined cobot assignment and job shop scheduling problem. (section 4.3)

5. In this thesis, a numerical study has been done on a real-world data set. It is shown that the hybrid genetic algorithm outperforms the constraint programming on the complex real-world problem. Additionally, the impact of a specified number of cobots on the production environment is shown. (section 4.5.3)

6. A numerical study is done on the literature data set. It is shown that the hybrid genetic algorithm can compete with state-of-the-art algorithms from the literature. (section 4.5.4)

1.6.2 Contribution B

When a metaheuristic, like a genetic algorithm, generates a solution for a scheduling problem, it is unclear what it looks like. Process mining algorithms are built to visualize and analyze existing business processes. Therefore, an event log file of the process is generated, and a process mining algorithm is applied to this event log file. Since this has not been done before, a connection between metaheuristics and process mining algorithms is developed. A tool

has been built to analyze metaheuristic solutions and find bottlenecks where a cobot can be deployed. This is described in detail in Chapter 5. The focus of this chapter is to answer research question **RQ B, B1, B2** and the contribution is the following:

1. Section 5.2 combines metaheuristics with process mining. Metaheuristic solutions are stored as event log files. Process mining algorithms are applied to these event log files in the next step to generate a process visualization. These visualizations are manually analyzed.

2. A set of scenarios is created to analyze the impact of various problem-specific parameters on the cobot placement. It is shown how variable sizes and different colour patterns can be used for the solution analysis. Additionally, the problem-specific parameters are analyzed by the influence on the cobot placement. (Section 5.3)

1.6.3 Contribution C

In the final contribution of this work, a metaheuristic, specifically a memetic algorithm, is combined with process mining algorithms. During the run of the memetic algorithm, a subset of solutions is stored as an event log file and passed to the process mining algorithm. This algorithm reveals knowledge implicitly given in the solutions and passes this knowledge back to the memetic algorithm to boost its performance. A numerical study on different problems and datasets analyzes the algorithm's strengths and weaknesses. The problem files and algorithms code is published to allow the reproducibility of the results. Details regarding this algorithm can be found in Chapter 6. The focus of this chapter is to answer research question **RQ C, C1, C2, C3** and the contribution is the following:

1. A memetic algorithm is combined with a process mining algorithm. The memetic algorithm generates solutions, and the process mining algorithm learns from them and passes the extracted knowledge back to the memetic algorithm. (Section 6.2)

2. All problem files and the code to solve these problems has been published. This can be found in Section 6.3.1.

3. In a numerical study, problem instances from three data sets are solved with the combined algorithm and a constraint programming formulation. This shows how much the solution quality can be increased through a novel combination of metaheuristics and process mining. (Section 6.3)

1.7 Thesis structure

An overview regarding the state of research and related work of this thesis is given in Chapter 2. Section 2.1 focuses on scheduling problems and why they

are still relevant for modern industry. Section 2.2 discusses state-of-the-art algorithms used to solve a large set of problems. Section 2.3 shows related work in the process mining area. Section 2.4 highlights where and how metaheuristics are already combined with process mining algorithms.
Chapters 3 to 6 present the main part of this thesis. Each chapter stands for one paper and is prefaced with an evaluation of each author's contributions and further information regarding the publication. A preview of this publications is:

> **Alexander Kinast**, Karl F. Doerner, Stefanie Rinderle-Ma. "Biased random-key genetic algorithm for cobot assignment in an assembly/disassembly job shop scheduling problem." *Procedia Computer Science 180 (2021) 328–337*
>
> **Alexander Kinast**, Roland Braune, Karl F. Doerner, Stefanie Rinderle-Ma, Christian Weckenborg. "A hybrid metaheuristic solution approach for the cobot assignment and job shop scheduling problem." *Journal of Industrial Information Integration 28 (2022) 100350*
>
> **Alexander Kinast**, Karl F. Doerner, Stefanie Rinderle-Ma. "Combining metaheuristics and process mining: Improving cobot placement in a combined cobot assignment and job shop scheduling problem" *Procedia Computer Science 200 (2022) 1836–1845*
>
> **Alexander Kinast**, Roland Braune, Karl F. Doerner, Stefanie Rinderle-Ma. "Optimizing the Solution Quality of Metaheuristics through Process Mining based on Selected Problems from Operations Research." *BPM Forum (2023)*.

In Chapter 7, an overview of the results generated in the individual papers of this work is given. After discussing these results, this thesis concludes with an outlook and future possibilities.

2 Methodology and state of research

This chapter gives an overview of the current state of research regarding the background of all research questions. Since this work aims to combine process mining and metaheuristics, it is an interdisciplinary field of research. The methodological contribution of this thesis is, on the one hand, the newly developed encodings and, on the other hand, the improved algorithms. These are validated through large computational studies.
To explain the current state of research, an overview of state-of-the-art scheduling problems, metaheuristics, and process mining algorithms is given. This chapter concludes with a section that focuses on existing interactions between process mining algorithms and metaheuristics and a section regarding the visualization possibilities of process mining algorithms. While the previous chapter gave an overview of the aims and research questions of the work, this chapter focuses on the state-of-the-art and how this work extends the state-of-the-art in particular areas.

2.1 Scheduling problems

A simple optimization problem could be a two-dimensional field (10m x 10m) with a food source. One can imagine that birds will circle towards this food source in a cornfield. However, these simple ideas often do not work for higher dimensional spaces. Adding a third dimension with 10m will increase the search space from 100 square meters to 1000 cubic meters. Due to the ”curse of dimensionality”, the search space of a function grows exponentially regarding the dimension of the problem. [25]
Due to rapid technological advancement, more data than ever is available. This comes with new large-scale problems. Examples are logistic problems for the first kilometer-tall building or neural networks with a billion variables.[26]
To stay competitive, efficient production scheduling is essential for many companies. However, most scheduling problems are np hard, making them impossible to solve exactly. [27]
In production scheduling, tasks must be assigned to production resources so that all constraints are met and the objective function is optimized. A well-known problem in this field with a large number of real-world applications is the job shop scheduling problem (JSSP). Another problem in this field is the flexible job shop scheduling problem (FJSSP), which is even harder to solve than the JSSP due to its flexibility. [28]
Real-world problems are very diverse. Examples are the lighting industry [29], automotive industry [30], aerospace industry [31], or the registration for a university [32].
Industry 4.0 will shift production to online demands, real-time and reactive methods. To overcome problems introduced with this shift, *developments in the scheduling area are critical to success.* [33]

2.2 Metaheuristics

In [34], metaheuristics are described as *high-level problem-independent techniques that can be applied to a broad range of problems.* Additionally, the number of publications in the research area of metaheuristics over the last 30 years is analyzed in this article. The research area is rapidly growing each year, which has led to over 50% of the analyzed articles being published in the last four years.
Due to the ”no free lunch theorem”, there can not be one algorithm that outperforms other algorithms on all problems. Therefore, it is important to analyze how well a specific algorithm works for a specific optimization problem. Additionally, problem knowledge must be exploited to improve algorithms' performance on a specific problem. [35] In [36], it is described that biological, physical, or chemical processes inspire the most basic metaheuristics. In [37], the following examples for metaheuristics are given:

- Ant colony optimization
- Evolutionary computing
- Iterated local search
- Genetic algorithms
- Simulated annealing
- Tabu search

An example of this biological inspiration is a genetic algorithm. This algorithm tries to mimic biological evolution. [38]
An example of a physical inspiration is simulated annealing. This algorithm emulates the physical process of cooling down a solid. [14]
An algorithm based on a chemical process is ant colony optimization. Real ants search the neighbourhood of the nest for food. If they find food, they evaluate it and place chemical pheromones on the ground on the way back to the nest. This guides other ants and allows the swarm to find the shortest path to the food source. [39]
In scheduling problems, metaheuristics compete with fuzzy logic, expert systems, machine learning and constraint programming. [40] To improve the performance of the classical metaheuristics, these basic algorithms are combined with local search methods. A genetic algorithm, for example, excels at exploring the search space. However, it is not the best algorithm to exploit a certain area of the search space. Local search methods are then used to exploit promising regions in the search space. These combined algorithms are called memetic algorithms (MAs) and can be considered state-of-the-art algorithms for scheduling problems.

2.3 Process mining

Process mining techniques are based on data mining, computational intelligence, process modelling and analysis. The basis for these techniques is a so-called event log file. This file contains various information regarding a process, such as time stamps, used resources or organizational information. The techniques applied to this event log file are grouped into process discovery, conformance checking, and process enhancement.[19]
Process discovery techniques focus on knowledge extraction. Therefore, most of these methods mine a process model (Petri net, workflow net, ...) from a given event log. Conformance-checking methods start with a model and an event log file. These methods try to *detect deviations, locate and explain these deviations, and measure the severity of these deviations.* [41] All techniques that improve a given process based on information recorded in an event log file are called process enhancement. [42]
This thesis focuses on knowledge extraction through process discovery. Current state-of-the-art process discovery techniques are the inductive miner [43], the alpha miner (different variants) [44], the heuristic miner [45], the genetic miner [46], or the mining for local process models [47].
Most of these algorithms focus on creating start-to-end models which describe the whole process. In contrast, a rather new approach, the local process model miner, focuses on capturing frequent patterns that describe key parts of the process. [47]

2.4 Metaheuristics and process mining

Since a large number of process mining techniques have problems with noise in the data, a more robust algorithm can be helpful for noisy data. As stated in Section 2.2, metaheuristics, like a genetic algorithm, can be applied to a large number of problems without large adaptations, and genetic algorithms are robust and can resist errors. In [46], a genetic algorithm is used to mine process models for instances that could not be mined with traditional process mining approaches. The challenge for this algorithm is that mined models must be able to reproduce all behaviour captured in the log while not allowing too much additional behaviour.
In [48], simulated annealing is used to create a workflow net out of an event log file. In [49], a genetic algorithm is used to create a process tree out of an event log file. In [50], a genetic algorithm is used to generate a Pareto front of medical process models.
What can be seen is that quite some research has been done to apply metaheuristics to process mining problems. However, it has not been researched if and how process mining algorithms can be used to improve the performance of a genetic algorithm.

2.5 Process mining as visualization tool

Since process mining has drawn a lot of attention in previous years, many software tools are available. Process mining allows the visualization of processes where only log files exist. Open-source software such as ProM and PM4Py can create a process model from an event log file. Additionally, many commercial process mining tools exist, e.g. Celonis, Disco, or Signavio. [51]
An example of such a process visualization of a real-world process is described in [52]. In this paper, the emergency medical service system of a large area in Turkey is visualized with process mining. This area had 187248 emergency calls in 2016, and the process is visualized with the open-source software ProM.
In [53], process mining is used to build a predictive model for the establishment of maintenance inspection intervals.
In [54], process mining is used to analyze COVID-19 management for cancer patients.
Process discovery is a state-of-the-art method to visualize different real-world processes where an event log file exists. However, it is not researched yet whether process discovery can be used to visualise evaluations of metaheuristic algorithms.

3 Biased random-key genetic algorithm for cobot assignment in an assembly/disassembly job shop scheduling problem

Full publication details

Alexander Kinast, Karl F. Doerner, Stefanie Rinderle-Ma. "Biased random-key genetic algorithm for cobot assignment in an assembly/disassembly job shop scheduling problem". Procedia Computer Science. Volume 180, 2021, Pages 328-337.

Author contributions
I am the main author of this paper. Therefore, I am responsible for the following:

- Conceptualization
  Formulation of research goals and aims
- Conceptualization
  Development of methodology
- Software
  Implementation of computer code and algorithms
- Writing - Original draft & review
  Writing the original draft, review based on feedback and presentation at the conference

The two co-authors are my two supervisors, Karl F. Dörner (supervision, conceptualization, writing - review & editing) and Stefanie Rinderle-Ma (supervision, conceptualization, writing - review & editing).

Information on the Status
Submitted: 24.07.2020
Accepted: 09.09.2020
Published: 20.02.2021

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 180 (2021) 328–337

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Industry 4.0 and Smart Manufacturing

Biased random-key genetic algorithm for cobot assignment in an assembly/disassembly job shop scheduling problem

Alexander Kinast[a,*], Karl F. Doerner[b], Stefanie Rinderle-Ma[c]

[a]*Research Platform Data Science @ Uni Vienna, Währinger Straße 29, 1090 Vienna, Austria*
[b]*University of Vienna, Department of Business Decisions and Analytics, Oskar-Morgenstern-Platz 1,1090 Wien*
[c]*Workflow Systems and Technology, University of Vienna, Währinger Straße 29, 1090 Vienna, Austria*

Abstract

Nowadays many manufacturing companies try to improve the performance of their processes by including innovative available technologies such as collaborative robots. Collaborative robots are robots where no safety distance is necessary, through cooperation with human workers they can increase production speed. In this paper we consider the collaborative robot assignment combined with the job shop scheduling problem. To solve this problem, we propose a genetic algorithm with a biased random-key encoding. The objective function for the optimization is a weighted function that factors in production cost and makespan that should be minimized. We propose a special encoding of the solution: the assignment of cobots to workstations, the assignment of tasks to different workstations and the priority of tasks. The results show how much the weighted objective function can be decreased by the deployment of additional collaborative robots in a real-world production line. Additionally, the biased random-key encoded results are compared to typical integer encoded solution. With the biased random-key encoding, we were able to find better results than with the standard integer encoding.

Keywords: Genetic algorithm; Job shop scheduling; Biased random-key encoding; Collaborative robots

* Corresponding author.
E-mail address: alexander.kinast@univie.ac.at

1877-0509
10.1016/j.procs.2021.01.170

1. Introduction

In modern industry, fully automated robots are already frequently in use. Robots can repeat the same static task with high speed and precision, but they are not suitable for highly flexible productions. In such productions, human skills are used to get the desired flexibility.

In medium-sized companies the deployment of robots is often too expensive and repetitive tasks are done manually. A cheaper alternative to fully automated robots and pure manual work is human-robot collaborations. Collaborative robots (abbreviation: cobots) differ from traditional robots in the way that no safety distance is necessary. Since they are in direct contact with humans, they move more slowly compared to typical robots (around 0.5 - 1m/s in comparison to the 1.6m/s of a human actor). They can do some tasks on their own or in cooperation with a human actor, however since they move more slowly compared to a human, it is assumed that they are slower in executing tasks on their own as well. According to their manufacturers they can do jobs like pick and place, screw driving, injection molding and many more. A typical cooperative task would be, that a human actor places screws on a work piece and the cobot screws them 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, often on one workspace on the same workpiece/task, with a cooperative robot (cobot).

An example for a raising field of cobot applications is the end-of-life disassembly of electric vehicle batteries. They must be disassembled for recycling and have a high negative impact on the environment if they are disposed wrong. Disassembly is not that easy, since the battery contains substances that are hazardous to humans and it is necessary that the cells of the battery is not damaged in the disassembling process.

Additionally, these end-of-life disassembly tasks have unpredictable volumes and high variation due to the difference in car models. Robots are not applicable for the disassembling since there is so much variance in the battery types. Disassembling contains many steps that can be done by a cobot. Batteries are held together by many screws and unscrewing is a repetitive and uninteresting task for a human. In Fig. 1 it can be seen, that the cobot is placed in a way that it can interact with the human worker during the disassembly process. By working closely together with a human actor, cobots can reduce the costs and risks in this process [2].

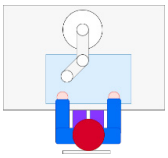


Fig. 1. Human-cobot interaction

Typically, in those production systems the tasks need to be assigned to specific workstations. This is an operative problem that needs to be solved to handle all incoming orders. In those workstations all resources that are necessary for production are required, however there is the additional tactical problem of the cobot assignment to speed up bottleneck workstations.

2. Problem description

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). A task is one production step that is necessary for the completion of one job, like mounting of a mechanical part or screwing in screws. Typically, a standard production time for such a task exists. An example for an optimization goal is, that the makespan (total production time until all jobs are finished) is minimized. However, lots of other metrics can be used for single objective optimization or as combination in a multiple objective optimization. In this paper, an idealized environment with no uncertainties and no stochastic influences is assumed. This means, the production of a task with an average duration x will always take x time units.

If a company wants to invest in cobots, it is likely that due to budget constraints cobots are only installed at those workstations where a high improvement can be reached. In those companies, the orders that should be produced over the next weeks/months are frequently already known. For these orders, the tasks should be scheduled and the cobots should be assigned to workstations where a high improvement can be reached. If a cobot is assigned to a workstation, it is not likely that the cobot will be redeployed. Typically, it is not easy to identify workstations, that are a bottleneck in the current planning period. The problem consists of an assignment problem of cobots to workstations and a job shop scheduling problem.

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 a lot. Workstations that can do the same tasks are grouped as workstation groups. To find the best suiting workstation in a workstation group for a task with a specific optimization goal, the classical job shop scheduling problem is therefore extended with a task to workstation assignment. [3]

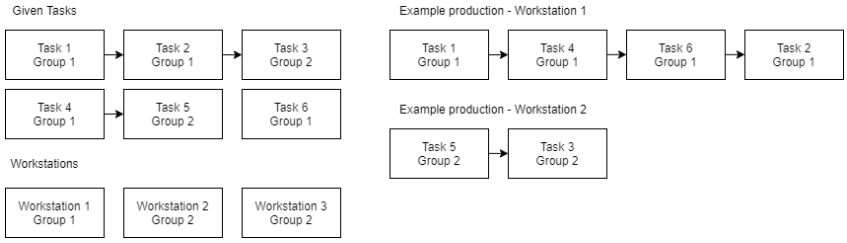


Fig. 2 Precedence graph

In Fig. 2 it can be seen that tasks might have precedence tasks that have to be executed before the task can be done. In this paper, it is assumed that required tasks have to be finished before a subsequent task can start producing. Fig. 2 shows, that Task 4 on workstation 1 has to be finished before Task 5 on workstation 2 can start. It is not necessary to produce all tasks of a job at one time, without tasks of other jobs in between. If multiple tasks can be produced at the same time on a workstation, they have to be arranged depending on their urgency. It might be preferable to produce a task with ten follow up tasks before a task that has no follow up tasks. Therefore, the classical job shop scheduling problem is extended with a priority assignment for each task in a job.

We extend the job shop scheduling problem with a cobot assignment and review a combined problem. Mixing this operative problem with a tactical cobot assignment makes sense, because in the literature it is often described, that a cobot can be redeployed to a new workstation within half a day. The combined problem should solve the following decisions:

- Since only a limited amount of cobots can be bought, on which workstation should these cobots be installed?
- 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 can be used, optimizing this combined cobot assignment and job shop scheduling problem. An example would be to minimize the production cost and opt for the highest profit. This will not be applicable for a real-world production, since only the cheapest workstation in a group will have more tasks assigned and all other workstations will be neglected. In real productions, a late delivery will often have various different negative consequences. To prevent those 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 produce in parallel. To find a good compromise solution, the objective function will be somewhere in the

middle. A weighted average of the production cost and the makespan. There might be solutions where the reduced production cost outweighs an increased makespan.

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. Our general weighted objective function can be described as follows:

$$Fitness = alpha * production\ cost + beta * makespan$$

3. Related work and research contribution

A lot of research has been done in the field of job shop scheduling since the late 1950s. Researchers tried to optimize the problem with different well know performance measures like makespan, mean flow time (average time a single job spends in the shop) or lateness of the jobs (how well due dates are respected). Various heuristics like tabu search, genetic algorithms and variable neighborhood search have been used to solve the problem with a single or with multiple objectives. Most of these papers are research papers that focus on method development and only around 8% address real-world industrial applications. [4]

Another research direction focuses on the influence of disruptions and rescheduling of processes. This is due to the fact, that real-world environments are never as deterministic as it is assumed in method development. Examples for such disruptions are machine breakdowns, operator illness, new priority jobs, cancelled jobs or changes of job deadlines. There are various methods, how to react to such disruptions.

One option would be full-reactive scheduling with priority rules. This means, that the decision which task is produced next is done locally on the machine. Each task gets a priority assigned based on machine and job attributes. Another option would be to schedule jobs in a more robust way, that even with machine breakdowns the objective function is influenced to the smallest extent possible. [5]

A different way to deal with uncertainties in production is the application of fuzzy sets for uncertain process such as processing time. Based on this fuzzy set a satisfaction grade for different objective values can be calculated. A metaheuristic like a genetic algorithm can then be applied for the optimization of the satisfaction grade. [6]

Some researchers even propose to apply deep reinforced learning for job shop scheduling problems. Therefore, an agent-based learning approach could be used. This agent-based learning approach has a state that describes the current situation of the environment and has actions it can take. When an action is taken, a positive or negative reward is received. This is called 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 currently be achieved with these deep reinforced learning approaches are nowhere near what can be achieved with metaheuristics. [7]

The research contribution of this paper can be divided into three parts. The first part is, that the job shop scheduling problem is extended with a tactical cobot assignment problem. The second part is, that we developed a solution approach for this combined problem. In this solution approach we adapt a biased random key encoding developed by Ribeiro to solve the combined job shop scheduling and cobot assignment problem. In the third part, a numerical study on a real-world data set shows that this new encoding works better than an existing encoding. Since the real-world data set is way bigger than many problems considered in the literature, runtime is already a huge limiting factor. Therefore, in the current version the evaluation of one solution is always deterministic. In future research, it could be interesting to see how the flexibility of cobots can be used to prevent bottlenecks on breakdowns or in a stochastic environment.

4. Solution method

4.1. Overview

The job-shop scheduling problem is an NP-hard problem. This means that larger instances cannot be solved exactly in reasonable time. In the literature various metaheuristics like genetic algorithms have been applied successfully on large instances. These metaheuristics are used to receive approximations to the global optimum of the problem specific objective function [8].

To solve the problem described in the previous chapter, hence, we decided to implement a genetic algorithm. This genetic algorithm starts by creating a randomly initialized population that is improved over the duration of the algorithm. To generate new solution, the algorithm uses the following steps until a stopping criterium is reached:

- **Evaluation:** Assigns a fitness value to every individual of the current solution. The fitness describes how good a solution is regarding the selected objective function.
- **Selection:** Selects individuals from the initial solution that act as parents for the next generation. Fitter individuals should have a higher chance of being selected as parent.
- **Crossover:** Takes two parents to create one or two new solutions for the next generation. Different crossover variations exist.
- **Mutation:** Changes a solution in a way, that new points in the solution space are discovered. This should prevent premature convergence of the algorithm.

The solution needs to be encoded in a way, that the operators can work with the representation. This means the operators have to be implemented for each representation.

The algorithm will stop once a termination criterium is reached. Examples are a maximum number of generations or the finding of an acceptable solution [9].

4.2. Encoding

To encode a solution, a biased random-key approach is used in the programming language C#. In this approach a solution representation is a vector of double values with the lower bound zero and the upper bound one.

This simple representation allows metaheuristics like a genetic algorithm to easily create new solutions. This solution encoding has already been used with success on several classical optimization problems (including job shop scheduling problems) as well as on real-world applications [10].

Each task can be produced on a group of workstations. For all tasks with a workstation group with more than one workstation, the workstation is encoded as double value. This double value is decoded during the evaluation. This can be seen in the grey fields in Fig. 3.

The second value that is 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. 3.

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 available workstations, that have no cobot assigned yet. This can be seen in the yellow field in Fig. 3.

For each field in Fig. 3, a C# double data type is used. This double represents a real number with 8 bytes memory. This means it has a precision of 15-17 digits. [11].

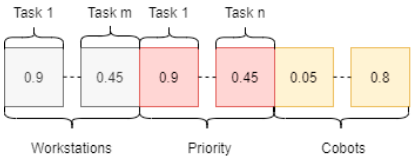


Fig. 3. Biased random-key encoding

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

4.3. Design Decisions

The framework that was used to implement the encoding and to run the genetic algorithm was HeuristicLab. HeuristicLab is a framework for heuristic and evolutionary algorithms that can be extended easily using a plugin-based architecture [12].

To get comparable results for the different encodings of one individual, the operators that are used in the genetic algorithm should be comparable between the two encodings.

Following operators can be implemented for integer and real value encoding:

- **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 the two parents. A parameter alpha defines how close the solution is either to parent one or to 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 alpha. For the integer encoded solution a rounded version is used.

During the development of the genetic algorithm, other operators like 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 with the operators described above.

4.4. Evaluation

To evaluate one solution, the real-world problem is modelled with all real-world constraints in a evaluation framework. Every time the genetic algorithm creates a new encoded solution (initialization and for every individual that is created using genetic operators) the evaluation method in the evaluation framework is used to create a fitness value. In Fig. 4 we can see that the encoded solution gets decoded and passed to the evaluation method. The evaluation method is deterministic and returns a fitness value to the optimization.

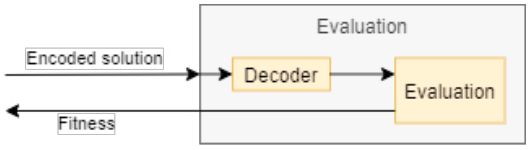


Fig. 4. Decoding and evaluation

The objective function that is used to receive a fitness is currently a weighted combination of the production cost and the makespan. Therefore, the general fitness function from chapter 2 is used with alpha = 1 and beta = 5.

5. Real world data set

The real-world data contains many typical elements of a job shop scheduling problem. The data contains orders that group together tasks. Tasks in that order have a fixed sequence of production, however it is possible to produce tasks of other orders in between. Each task can be produced on a group of workstations. The data contains the standard production time for each task and each workstation modifies this production time by a fixed factor. However, this means that the current version of the evaluation is fully deterministic. Additional environmental uncertainties can be included in later versions.

The dataset has the following metrics:

- 54 workstations
- 210 orders
- 1265 tasks

The data set is from the production of mechanical parts like engines, pumps and housings. The workstations are in the following areas:

- Heat treatment furnaces
- Prefabrication
- Assembly

In those production areas, workstations are grouped in workstation groups. Each workstation in such a group has individual production costs and production speed. Based on the number of produced products that a workstation is in use, there will be setup, de-setup and production costs/times. The costs and the production speed can vary across several workstations in one workstation group. For this paper it is assumed, that cobots can be installed on all workstations. In the real data set, some tasks have a workstation given. To increase the complexity of the data set, it is assumed, that a task can be produced on all workstations of that specific workstations group.

6. Preliminary results

The biased random-key and the integer encoding are both tested on the real-world data set. To increase the amount of different test cases, three different versions of the large real-world data are calculated. The first version is the full data set, the second and third version are each half and each quarter of the data evaluated independently. The data is split by splitting the jobs. In all versions five cobots are assigned to workstations by the genetic algorithm. Based on [1], it is assumed, that a cobot will increase the production speed of a workstation by 30% which will also lead to a cost reduction of 30%. The data sets in Table 1 and Table 2 are named based on the following schema:

- “Item set identifier”_”Minimum job”-“Maximum job”

An example for this naming is “I2_1-632” which means, the unique identifier for the data set is I2 and the jobs 1 to 632 are used.

This leads to a total of 7 different test sets. For each test set, the biased random-key and the integer encoding were run ten times with 100, 200 and 500 generations. Due to runtime limitations, it was not possible to perform more test runs, but more test runs would have increased the quality of the solution. For the following results the integer-based encoding has the abbreviation Int and the biased random-key encoding has the abbreviation Real. By running a simple rule-based solution, we found out that the production costs are around 5 times as high as the makespan. To weight production costs and makespan in a way, that they influence the solution equally, the general objective function described in chapter 2 was used with the following parameters:

- alpha = 1
- beta = 5

The genetic algorithm had the following other properties:

- Mutation rate: 5%
- Elite Solutions: 1
- Individual per generation: 100
- Minimization: true

The value that is received from the objective function is the fitness value that is assigned to a specific solution. Since we want to minimize the objective function, fitter individuals have a lower value assigned.

Table 1. Computational results, with cobots assigned (I1 = full data, I2-I3 = two halves, I4 … I7 = four quarters)

Encoding – generations	Int – 100	Real – 100	Int – 200	Real – 200	Int – 500	Real – 500
I1_1-1265	59629580	62130983	59863431	59474958	59399037	55755062
I2_1-632	18813579	17629346	18858999	17349055	18809609	16829534
I3_633-1265	41346574	42832277	41503659	40100420	40111042	37362107
I4_1-316	8574542	7479014	8402628	7042577	8399473	6820375
I5_317-632	10669668	9987885	10602398	9881887	10544902	9423611
I6_633-948	14710704	14020393	14749659	13349247	14715838	12871322
I7_949-1265	25528767	26414212	24867220	25632205	25174964	22255353
Average	25610488	25784873	25549713	24690050	25307838	23045338

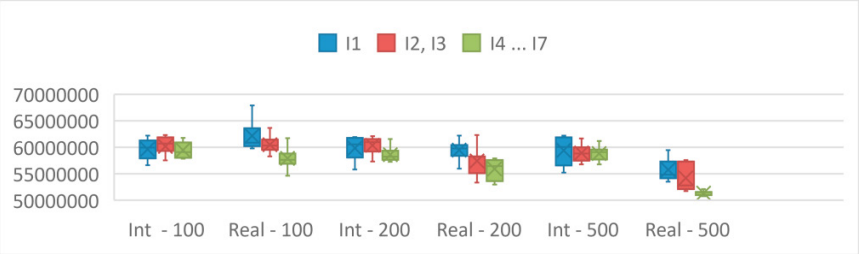


Fig. 5. Computational results with standard deviation

In Table 1 the average results of the test runs can be seen. What we can see on the values is, that using one half/quarter of the tasks does not mean that the fitness gets halved/quartered. The values for the halves/quarters are summed up, this means the “I2, I3” is equal to the full data with one cobot relocation after 50% of the tasks have been produced. The four quarters are similar to the full data set with three cobot relocations after 25%, 50% and 75% of the tasks have been produced. In Fig. 5 the value ranges of the individual runs of Table 1 with standard deviation is reported.

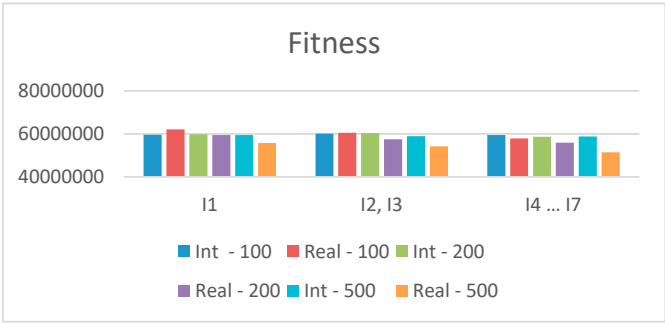


Fig. 6. Aggregated results

In Fig. 6 the data from Table 1 is visualized. For a better overview over the data, the split data I2-I3 and I4-I7 is aggregated. It can be seen, that by using the biased random-key encoding, the genetic algorithm was able to find better results in nearly all cases.

This biased random-key encoding is now used to compare the solutions with solutions where no cobot is assigned. This should give an idea, how much improvement can be reached by deploying the cobots to this real-world production environment. Therefore, the genetic algorithm is run for 100, 200, 500 generations for all data sets with no cobots being assigned to workstations.

Table 2 Computational results, real encoding, five cobots vs. no cobots assigned (I1 = full data, I2-I3 = two halves, I4 … I7 = four quarters)

Encoding – generations	No cobot – 100	Cobot – 100	No cobot – 200	Cobot – 200	No cobot – 500	Cobot – 500
I1_1-1265	71750212	62130983	70220400	59474958	68565491	55755062
I2_1-632	20185858	17629346	19928594	17349055	19344987	16829534
I3_633-1265	48172910	42832277	47290606	40100420	46063695	37362107
I4_1-316	8210794	7479014	8028540	7042577	7749723	6820375
I5_317-632	11647751	9987885	11513617	9881887	11231855	9423611
I6_633-948	16434327	14020393	16185096	13349247	15852957	12871322
I7_949-1265	30737913	26414212	29742436	25632205	29011880	22255353
Average	29591395	25784873	28987041	24690050	28260084	23045338

In Table 2 we see the computational results of the runs with and without cobots assigned to workstations. In the results with cobots, the genetic algorithm selects the workstations, where the five cobots should be deployed. The amount of generations run, changes the improvement that can be reached by deploying the cobots:

- 100 Generations: 13%
- 200 Generations: 15%
- 500 Generations: 18.5%

The more generations are evaluated, the better the solution is. After 500 generations an improvement of 18.5% is reached over the version without cobots. This means, the genetic algorithm makes better use of the cobots after more generations. The average improvement that can be reached over all three generation values is 15.4%.

When these results have to be applied to a real-world scenario, they have to be discussed with experts. Since there might be more limitations, that are not considered yet in the evaluation.

7. Conclusions and outlook

The real-world data set that was used is a representative data set for medium sized job shop scheduling problems. The only real limitation is, that the real-world production system allows the deployment of cobots. 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, algorithms like the NSGA-II could be used to optimize multiple objective values and generate a Pareto optimal front (solutions that are not dominated by other solutions) that are presented to a domain expert.

Both encodings that were used in this paper should be able to solve large real-world data sets. However, both offer the possibility to implement various additional encoding specific genetic operators. In future research, the effect of these different operators on the solution quality should be reviewed. Therefore, numerous evaluations have to be executed.

Additionally, it would be interesting to investigate, how much improvement can be reached by deploying different amounts of cobots in multiple real-world scenarios. The scenario could be tested with an increasing amount of cobots,

starting with one cobot. Adding cobots to a scenario has diminishing returns, since the first cobot can be deployed at the workstation where the biggest target function improvement can be reached. Further cobots are than deployed to workstations with decreasing amounts of impact on the objective function. This would make it possible to determine the ideal amount of cobots that should be deployed in each production system.

Acknowledgements

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 thankful for the real-world data set that was provided in cooperation with an industry partner.

References

[1] C. Weckenborg, K. Kieckhäfer, C. Müller, M. Grunewald and T. S. Spengler, “Balancing of assembly lines with collaborative robots,” *Business Research,* vol. 13, pp. 93-132, April 2020.

[2] K. Wegener, W. H. Chen, F. Dietrich, K. Dröder and S. Kara, “Robot Assisted Disassembly for the Recycling of Electric Vehicle Batteries,” in *Procedia CIRP 29 (2015) 716 – 721*, 2015.

[3] N.Sridhar, M. Victor Raj and K.Chandra sekar, “Minimizing Manufacturing Cost In Flexible Job-Shop,” *International Journal of Artificial Intelligence and Mechatronics,* vol. 1, no. 5, p. 2320 – 5121, 2013.

[4] B. Calis and S. Bulkan, “A research survey: review of AI solution strategies of job shop,” *Journal of Intelligent Manufacturing,* vol. 26, 2013.

[5] D. Quelhadj and P. Sanja, “A Survey of Dynamic Scheduling in Manufacturing Systems,” *Journal of Scheduling,* vol. 12, pp. 417-431, 2009.

[6] Fayad, Carole; Petrovic, Sanja;, “A Genetic Algorithm for the Real-World Fuzzy Job Shop Scheduling,” in *Lecture Notes in Computer Science*, Nottingham, 2005.

[7] B. Cunha, A. M. Madureira, B. Fonseca and D. Coelho, “Deep Reinforcement Learning as a Job,” in *International Conference on Hybrid Intelligent Systems*, 2020.

[8] B. Chen, C. N. Potts and W. Gerhard J., A Review of Machine Scheduling:, Kluwer Academic Publishers, 1998.

[9] M. Affenzeller, S. Wagner, S. Winkler and 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.

[10] M. L. Lucena, C. E. Andrade, M. G. C. Resende and 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.

[11] “Microsoft Docs,” Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/floating-point-numeric-types. [Accessed 21 09 2020].

[12] S. Wagner, G. Kronberger, A. Beham, M. Kommenda, A. Scheibenpflug, E. Pitzer, S. Vonolfen, M. Kofler, S. Winkler, V. Dorfer and M. Affenzeller, “Architecture and Design of the HeuristicLab Optimization Environment,” in *Advanced Methods and Applications in Computational Intelligence*, Springer, 2014, pp. 197--261.

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

Full publication details

Alexander Kinast, Roland Braune, Karl F. Doerner, Stefanie Rinderle-Ma, Christian Weckenborg. ”A hybrid metaheuristic solution approach for the cobot assignment and job shop scheduling problem”. Journal of Industrial Information Integration. Volume 28. 2022.

Author contributions
I am the main author of this paper. Therefore, I am responsible for the following:

- Data curation
  Maintain research data
- Conceptualization
  Formulation of research goals and aims
- Conceptualization
  Development of methodology
- Software
  Implementation of computer code and algorithms
- Writing - original draft & review
  Writing the original draft, review based on feedback and presentation at the conference
- Visualization
  Visualization of generated data

The co-authors are Roland Braune (software, formal analysis, validation, visualization, writing - draft & review), Karl F. Doerner (methodology, supervision, writing - review), Stefanie Rinderle-Ma (methodology, supervision, writing - review), and Christian Weckenborg (data curation, writing review).

Information on the Status
Submitted: 29.03.2021
Revision 1: 07.09.2021
Revision 2: 28.10.2021
Accepted: 21.04.2022
Published: 26.04.2022

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

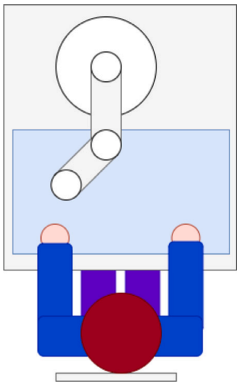


Fig. 1. Human-cobot interaction.

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:

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

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

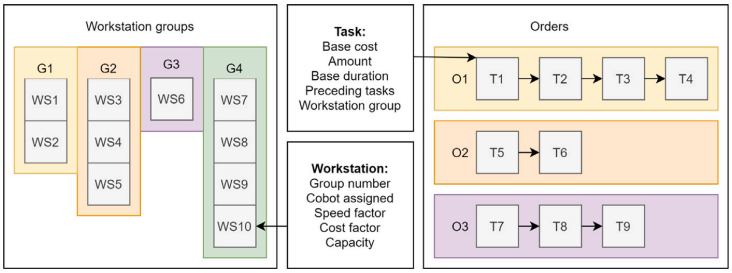


Fig. 2. Real-world problem overview.

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)

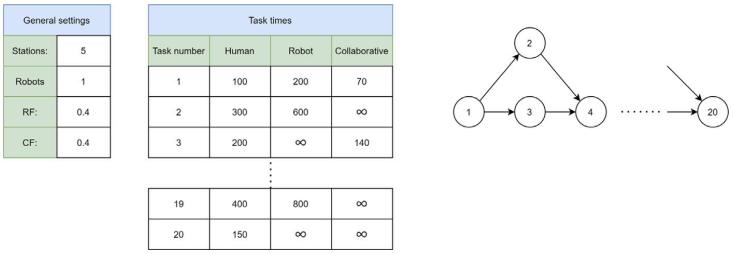
Table 3
Parameters used in the CP formulation.

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

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.

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

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

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].

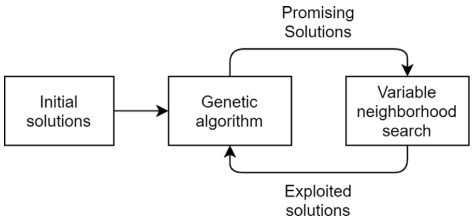


Fig. 4. Overview hybrid genetic algorithm.

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.

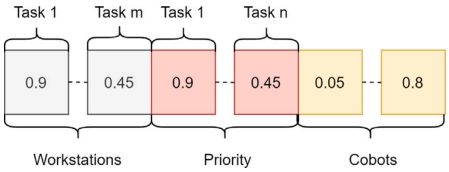


Fig. 5. Biased random-key encoding.

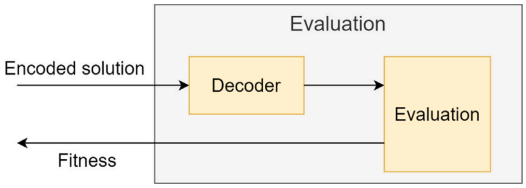


Fig. 6. Decoding and evaluation.

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

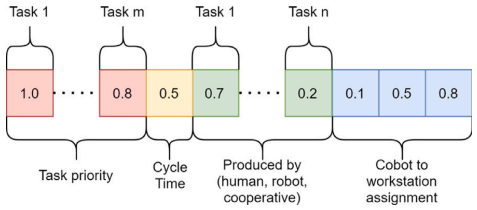


Fig. 7. Biased random-key encoding - Literature data set.

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]

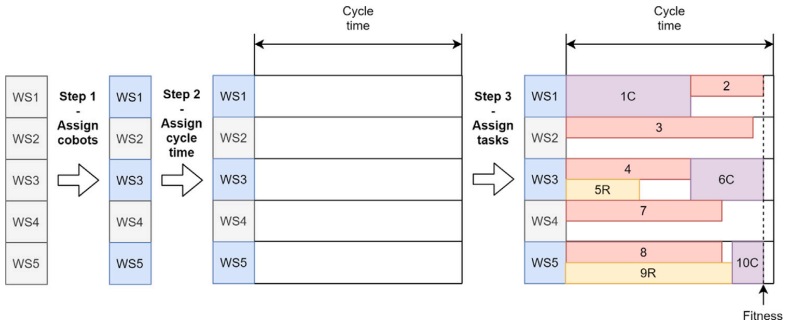


Fig. 8. Biased random-key encoding - Visualize steps.

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

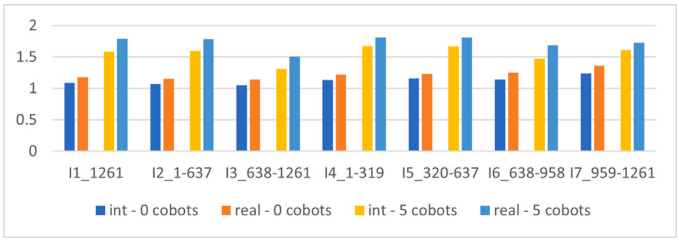


Fig. 9. Average solution quality - GA.

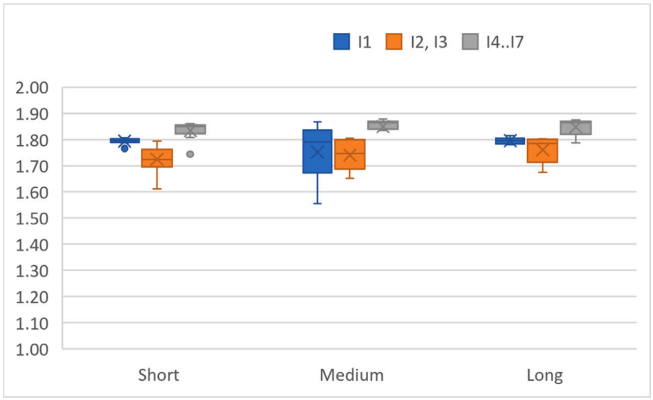


Fig. 10. Average results with standard deviation - Hybrid genetic algorithm - 5 Cobots.

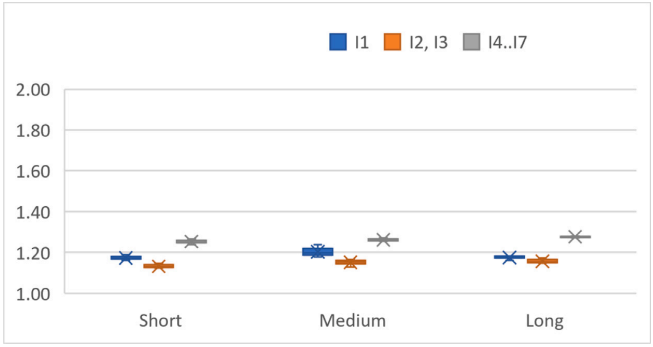


Fig. 11. Average results with standard deviation - Hybrid genetic algorithm - 0 Cobots.

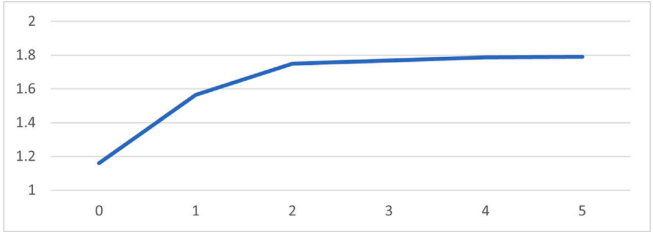


Fig. 12. I2_1-637 - Normalized fitness per cobot.

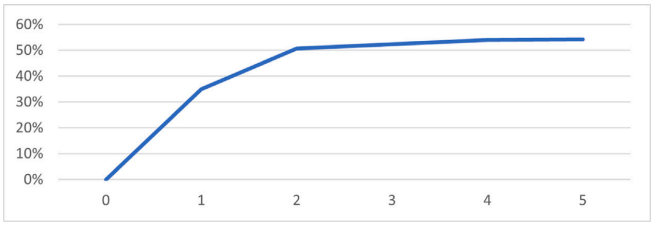


Fig. 13. I2_1-637 - Percentage improvement per cobot.

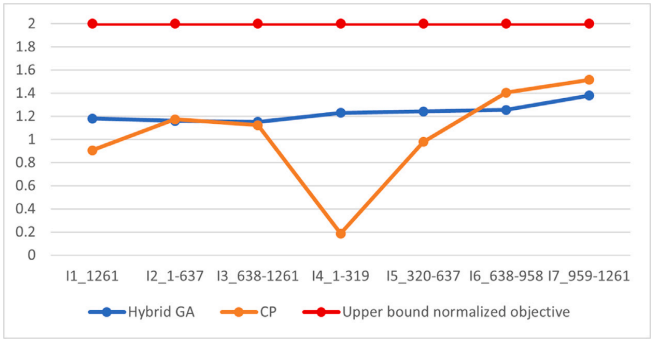


Fig. 14. Comparison CP/Hybrid GA - 0 cobots.

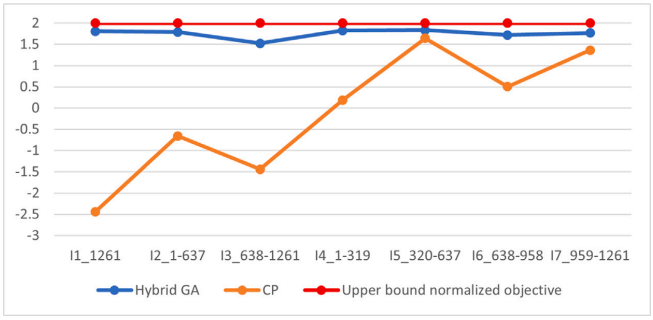


Fig. 15. Comparison CP/Hybrid GA - 5 cobots.

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.

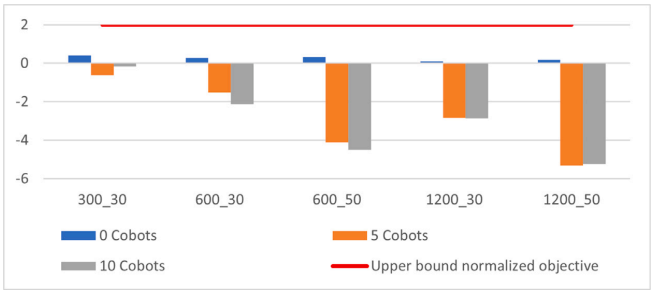


Fig. 16. Comparison - Artificial instances - CP model.

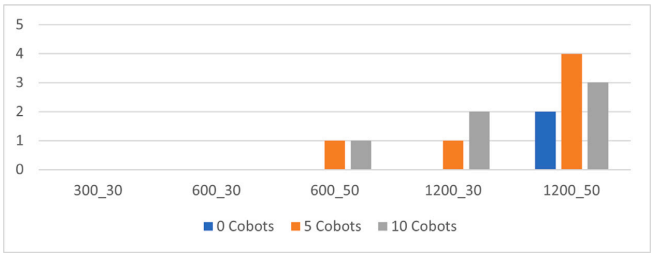


Fig. 17. Comparison - Artificial instances - No CP solution found (number of occurrences).

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

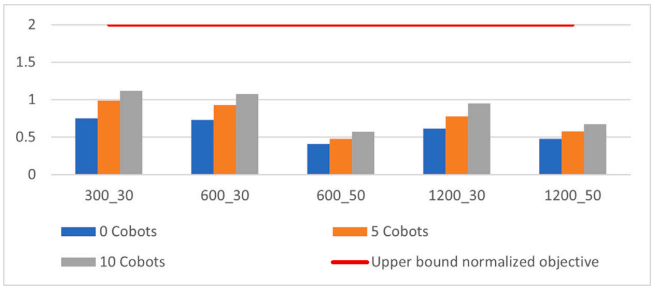


Fig. 18. Comparison - Artificial instances - hybrid genetic algorithm.

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:

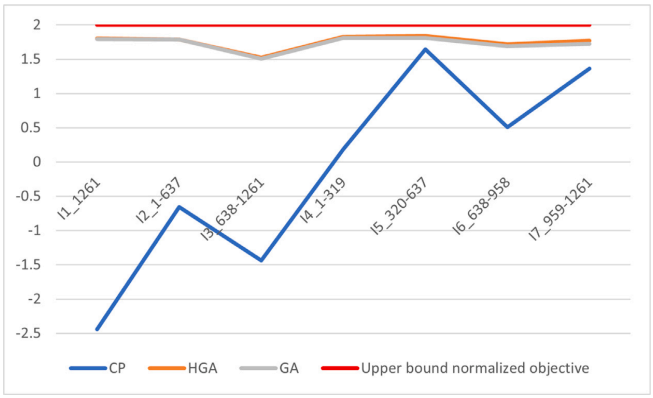


Fig. 19. Comparison GA/HGA/CP - 5 cobots.

- **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

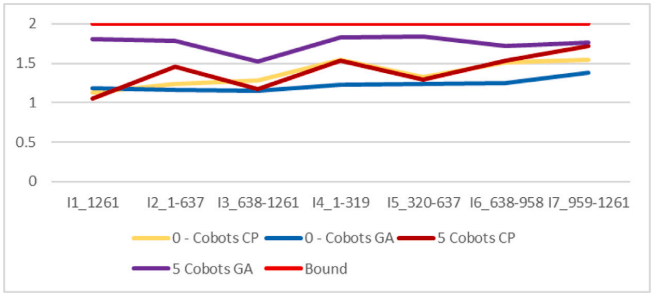


Fig. 20. Hybrid GA vs. CP (long run): normalized objective function values for real instances.

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.

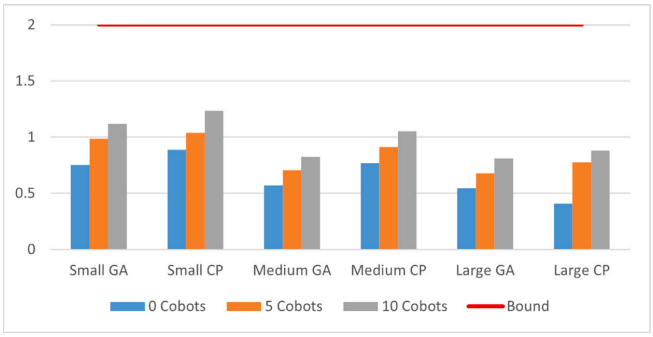


Fig. 21. Hybrid GA vs. CP (long run) - normalized objective function values for artificial instances.

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.

5 Combining metaheuristics and process mining: Improving cobot placement in a combined cobot assignment and job shop scheduling problem

Full publication details

Alexander Kinast, Karl F. Doerner, Stefanie Rinderle-Ma. "Combining metaheuristics and process mining: Improving cobot placement in a combined cobot assignment and job shop scheduling problem". Procedia Computer Science. Volume 200, 2022, Pages 1836-1845.

Author contributions
I am the main author of this paper. Therefore, I am responsible for the following:

- Conceptualization
  Formulation of research goals and aims
- Conceptualization
  Development of methodology
- Software
  Implementation of computer code and algorithms
- Visualization
  Creating visualizations for the data
- Writing - Original draft & review
  Writing the original draft, review based on feedback and presentation at the conference

The two co-authors are my two supervisors, Karl F. Dörner (supervision, conceptualization, writing - review & editing) and Stefanie Rinderle-Ma (supervision, conceptualization, writing - review & editing).

Information on the Status
Submitted: 06.08.2021
Accepted: 30.08.2021
Published: 08.03.2022

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 200 (2022) 1836–1845

Procedia Computer Science

www.elsevier.com/locate/procedia

3rd International Conference on Industry 4.0 and Smart Manufacturing

Combining metaheuristics and process mining: Improving cobot placement in a combined cobot assignment and job shop scheduling problem

Alexander Kinast[a,*], Karl F. Doerner[b], Stefanie Rinderle-Ma[c]

[a] *University of Vienna, Forschungsplattform Data Science, Kolingasse 14-16, 1090 Wien, Austria, alexander.kinast@univie.ac.at*
[b] *University of Vienna, Department of Business Decisions and Analytics, Oskar-Morgenstern-Platz 1,1090 Wien, Austria, karl.doerner@univie.ac.at*
[c] *Technical University of Munich, Department of Informatics, Chair for Information Systems and Business Process Management, Boltzmannstrasse 3, 85748 Garching, Germany, stefanie.rinderle-ma@tum.de*

Abstract

Human workers can share a workspace with modern collaborative robots (cobots). The main differences to traditional robots are, that workers do not need a safety distance when interacting with cobots, as they move slower than typical industrial robots. Cobots also have fast setup times compared to traditional resources. The hybridization is based on a previous work of the authors, where a job shop scheduling problem that is extended with a robot to workstation assignment with a hybrid genetic algorithm is solved. In this paper, the potential of hybridizing an optimization algorithm with process mining techniques to improve the solution quality by gaining information on the solution structure is analyzed. This additional information should help to guide the search process. Process mining techniques are presented to analyze the solutions and learn from them. The idea of this work is to understand the solutions generated by the genetic algorithms as process executions, e.g., the production of a part as a process instance executed across the selected work stations. Then, by generating process event log data out of selected solutions, state-of-the-art process mining techniques can be used as visualization and scanning tools for the underlying processes. This way, for example, bottleneck workstations in the production process can be highlighted. Based on created scenarios, this paper demonstrates how genetic algorithms and process mining techniques can be combined. In the future, it is planned that information that is mined from generated logs of an evaluation framework is used to improve the performance of hybrid genetic algorithms by using this information in a feedback loop. Generated insight in cobot placement can also be used for prescriptive analytics in real-world manufacturing companies that want to utilize cobots. The focus of this paper lies in the discussion of the usage of potential information extracted from process mining.

Keywords: Process mining; Hybrid genetic algorithm; Collaborative robots; Job shop scheduling; Prescriptive analytics

* Corresponding author.
E-mail address: alexander.kinast@univie.ac.at

1877-0509
10.1016/j.procs.2022.01.384

1. Introduction

Automation has gained significant importance in the past years, see e.g. [5]. Process automation is used to improve product quality, efficiency, and availability of the production process. For many processes in production companies, it is important to use state-of-the-art technology in order to stay competitive in the future. In [16], it is shown, that collaborative robots (cobots) offer new opportunities for different fields of manufacturing companies. These cobots are designed to work with human workers and no safety distance is necessary. Through this interaction with humans, cobots allow automatizing workstations, where it was previously not possible. Cobots can be used for activities such as picking, welding, assembling, or inspecting products.
The different cooperation modes between a cobot and a human actor are described in [3]. The mode that is the furthest away from cooperation is a traditional robot in a cell that acts alone. When a cobot and a human worker share a workspace, the following modes are differentiated:

- Synchronization:
  Even if the human and the cobot share a workspace in the synchronized mode, only one of both actors works on the task at the same time.
- Cooperation:
  In the cooperation mode, both actors can work simultaneously at the same shared workspace on different tasks.
- Collaboration:
  The most interactive mode is the collaboration mode, where both actors can work at the same time on the same task.

An example for that is interactive process automation based on a picking and placing station [9].
Cobots can be used if traditional robots are not flexible enough or if the workstation does not allow a degree of automation that is typically used for industrial robots [20]. Through a combination of the humans' strengths such as flexibility, adaptability, or decision making and the cobots' strengths such as speed, endurance, and accuracy the performance of many workstations can be increased. In comparison to traditional resources, cobots have convenient setup times (programmable within half a day) which allows companies to adapt to fast changes in production.
When deploying cobots to a real-world environment it is possible to use prescriptive analytics. In [8], it is described that prescriptive analytics uses historical and real-time data in combination with optimization algorithms and expert systems to predict possible outcomes of a system. This is done to predict delays or bottlenecks and present options for how these can be avoided. However, this means that it is necessary to understand why bottlenecks or delays happened in the past and in the case of cobots, why cobots are placed at specific workstations.

1.1. Problem description and Solution technique

Due to a budget constraint, most companies will only invest in a limited amount of cobots. These cobots should assist workstations, where a bottleneck is expected in the next planning period. Depending on the objective function, this can be workstations that will delay the reset of the production or workstations where the production cost can be significantly reduced by deploying a cobot.
When knowing the orders for the next planning period, it is important to determine on which workstation tasks of these orders should be produced and where to place cobots to have maximum impact. This impact can be measured in various ways, examples include production cost, makespan, lead time, tardiness, or a combination of these and other factors.
Genetic algorithms are commonly used for the optimization of complex problems. These genetic algorithms are population-based algorithms that start with an initial random generation. With genetic operators such as selection, crossover, and mutation, new generations get created until a stopping criterion gets reached. A big advantage of genetic algorithms is that they can be applied to nearly all optimization problems and deliver high-quality solutions. The basic concepts and properties of genetic algorithms are explained in [2].
Since traditional resources are not as flexible as cobots, most existing algorithms are not designed to reallocate resources. In our previous work [6], a new encoding for a job shop scheduling problem for a genetic algorithm is developed. This encoding is able to assign tasks to workstations, give them a priority, and assign cobots to worksta-

tions. It is shown, that even a small number of cobots can lead to a high improvement of the solution quality. This is due to the fact, that the cobots are deployed to bottleneck workstations that slow down the whole production or workstations that create high production costs.
A genetic algorithm, as described in [6] can be used. This algorithm considers the production speed and production cost from the different workstations in addition to the orders that should be produced on each workstation. Using this algorithm results in a high-quality solution, however, it might not be clear what factors or problem characteristics lead to the decisions of the algorithm.
Whenever the genetic algorithm generates an encoded solution, the evaluation framework is used to evaluate this solution. During this evaluation, the production of all orders is simulated and tasks are assigned to workstations. This task to workstation assignment with start/end timestamps and additional properties is logged to the event log file.
Process mining is a technique that can be used to mine implicit knowledge from log files. When the genetic algorithm is executed, log files from selected solutions can be created which then can be analyzed with process mining techniques and the results can be visualized.
Based on these considerations the following questions arise:

- What factors influence the cobot to workstation assignment?
- Can process mining visualize relevant factors that lead to the decision of the genetic algorithm?
- Can information gained from process mining improve the quality of results found by the genetic algorithm?

Since this is a novel optimization problem, it is not clear what factors besides the objective function are relevant factors that influence the placement of a cobot. Examples of such factors could be the number of workstations that can handle the same tasks, the duration of the individual tasks, or the number of tasks.
State-of-the-art process mining techniques should be used to visualize the attributes such as production cost and production time that lead to the decisions of the genetic algorithm. If process mining techniques can be used to visualize the decisions of the genetic algorithm, it might be possible to use this implicit knowledge to improve the performance of the genetic algorithm.

2. Solution method

2.1. Process mining

"Process automation and process mining are regarded as key technologies for digital transformation" [11]. Process mining has created high expectations due to the increased level of transparency [10], particularly in manufacturing [14]. Process execution information – explicitly managed by a process engine or implicitly executed by one or several information systems – is stored in process event logs. The main part of a log is a large number of traces. A trace is a set of events related to one case. In a production company, this trace could be one order and the events could then define the production steps that have been executed to produce this order. Various extensions can be used to define attributes of traces and events. These attributes that are defined by extensions are essential for many process mining algorithms. The ".xes" format is commonly used for representing process event logs [1] and is also used for generated log files in this paper.
Process mining [15] comprises techniques to a) discover the underlying process model from the process event logs (process discovery), b) check if the process event log conforms to an expected process model (conformance checking), and c) enhance process models. To discover process models out of these event logs, well-known process mining techniques, such as the inductive miner can be used. In [13] it is shown, that the inductive miner is able to generate process models out of car manufacturing processes and identify bottlenecks in the manufacturing process. This inductive miner is used to generate BPMN models out of log data in this paper.

2.2. Design Choices

In order to use process mining to find factors that influence the placement of cobots, process mining has to be able to analyze the results generated by a metaheuristic. Process mining is based on implicit knowledge that is mined from event log files, however, this is not the default output of a metaheuristic. The genetic algorithm with a biased random-key encoding developed in [6], has been used to solve the extended cobot placement and job shop scheduling problem. The objective function is the sum of production cost and makespan. The genetic algorithm is extended in such a way, that a ".xes" event log file is created for the best solution that has been found during the execution of the algorithm. This can be seen in Figure 1.

The created event log file of the best solution of the genetic algorithm is used as a basis for further process mining

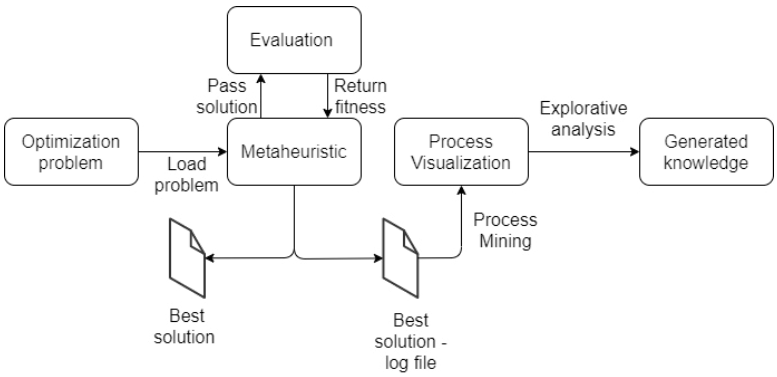


Fig. 1. Overall approach

steps. Potential results or typical patterns of this process mining step are visualized in this paper and explorative analysis steps are used to generated knowledge. This can be seen in Figure 1. The idea is, that the logs of the best solutions contain the implicit knowledge of why and based on what factors cobots are placed on workstations. In

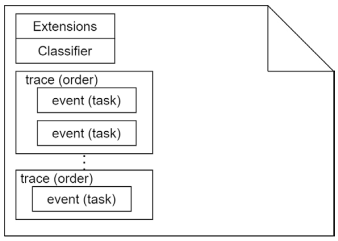


Fig. 2. Event log file structure

Figure 2, the structure of such an event log file can be seen. The file starts with an extensions section, that defines attributes that can be used for different process mining algorithms. Examples of such commonly used attributes are process start and end dates. The next section is a classifier section. These classifiers are predefined sets of attributes that are used for the nodes in the mined models. To reach the goal of displaying interesting attributes of workstations, the classifier for the scenarios is the workstation where a process is executed. The main part of the file contains traces and events. In this case, traces relate to orders that should be produced and events in these traces contain information about tasks that have to be performed. An example order could be the production of a mechanical part. To produce this mechanical part, different tasks such as cutting, drilling, welding, and assembling have to be done.

The standard representation for business processes is the Business Process Modeling and Notation (BPMN) format (www.bpmn.org). This format has the advantage of being easily understandable for end-users. Hence, we opt for mining process models in BPMN format from the process event logs. To mine a BPMN model out of the generated

event log files a process discovery algorithm, an inductive miner, as described in [7], is used. This process mining method generates a Petri net, which is converted to a BPMN model.
For analyzing the solution of the genetic algorithm mined BPMN model should be enriched with attributes that highlight why a cobot is placed on a specific workstation. Following [4], the attribute values are displayed by visualizations styles applied to the process tasks, i.e., the color and size of the nodes are varied in order to represent attribute values. The attributes that are visualized are attributes that have been logged to the event log file. The goal of this visualization is, that decision-relevant attributes for cobot placement are discovered, that can be used alongside mined information to improve cobot placement.

2.3. Implementation

The genetic algorithm with a biased random-key encoding as described in [6] is implemented in HeuristicLab [17]. For the evaluation of one solution, the simulation framework Easy4Sim provided by the RISC Software GmbH has been used. This framework can be easily extended and allows an efficient evaluation of individuals that have been generated by the genetic algorithm [19].
To perform process mining tasks on the event log files that have been created during the evaluation in the evaluation framework, the process mining framework pm4py is used [18].

3. Preliminary results

In [12], fundamental patterns that can occur when describing a business process are described. To find out which attributes influence the placements of cobots in the extended cobot assignment and job shop scheduling problem, scenarios are created with the following control-flow patterns:

- Sequence
  An activity can be done after a preceding activity finishes.
- Exclusive choice
  A branch is split into two or more branches. However, only one of these branches is active after the split.
- Simple merge
  Two or more branches are joined together. If one of the incoming branches is active, the outgoing branch will be active.
- Loop
  A loop allows the representation of cycles in a process model.

In the extended cobot assignment and job shop scheduling problem described in [6], tasks are limited to one predecessor and one successor task. This is also assumed in the generated scenarios and therefore no parallel split and parallel merge scenarios are created. Based on these patterns, the goal was to find cobot placement relevant attributes. In the created scenarios, each task that should be produced has a base duration that is modified by a speed factor of a workstation. When a task with a base duration of 100 seconds is produced on a workstation with a speed factor of 0.8, the task will be finished in 80 seconds. The cost to produce a task is the total time multiplied by the cost factor. This means if the task with a total duration of 80 is produced on a workstation with a cost factor of 1.5, a cost of 120 will be generated. The objective function for the optimization for these examples is simply the sum of costs and makespan (total time until all tasks have been finished). It is assumed that a cobot speeds up the production by 30% and, therefore, also reduces the costs by 30%.

3.1. Attribute visualization

In [4] it is described, how scaling, positioning, and labeling can be used to visualize multiple process attributes. For the following scenarios we use scaling, labeling, and additionally coloring of the nodes for the visualization. The

production cost will be visualized with the use of color in the BPMN. The color ranges from green (cheap workstation compared to the other workstations) to red (expensive workstations). The width of the nodes in the BPMN represents the production time of the workstations (a higher production time means a wider node). The production cost and production time are also displayed in the label of the node. At the left top of each scenario, the base duration of the tasks is stated. The color of the start and end node has no semantics.

3.2. Scenario 1 - exclusive choice

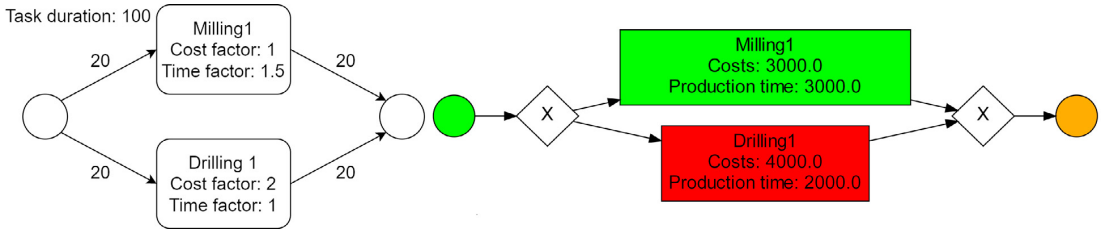


Fig. 3. Scenario 1 (Cost: color, Production time: width of the nodes)

In Figure 3 on the left side of the first scenario, two workstations work in parallel and both should produce 20 tasks that have a base duration of 100. For this parallel part, the tasks are split with an exclusive choice and merged with a simple merge after the production. On the right side, it can be seen that the mined BPMN model has been enriched with information from the workstations. It is shown that the workstation ”Milling1” created costs of 3000 and had a production time of 3000. The drilling workstation finished significantly faster in 2000 and had costs of 4000. When taking a closer look at how much the objective function gets influenced due to the deployment of a cobot, the following changes can be seen:

- Milling1
  - Cost: -900
  - Production time: -900
- Drilling1
  - Cost: -1200
  - Production time: -600 (not relevant for the makespan)

The total objective function in this example is 10000 (7000 costs + 3000 makespan). Since the production time of the drilling workstations is not relevant for the objective function, the algorithm will place a cobot to the milling workstation if one cobot is available, reducing the objective value to 8200.

3.3. Scenario 2 - exclusive choice and sequence

In Figure 4, a more complex version of the first scenario is shown. In comparison to Figure 3, the additional workflow pattern sequence is used. In the scenario depicted in Fig. 4 on the left side, there is only 1 task for milling/drilling and 1 task for assembling/turning. This task has a base duration of 2000. When running the algorithm with no cobots, the result that is shown at the bottom is generated. When allowing the algorithm to place a cobot, this cobot gets placed on one of the top workstations. This is due to the fact that these workstations have the highest costs and are relevant for the makespan. Placing the cobot on one of these workstations will reduce the objective function from 22000 (14000 costs + 8000 makespan in the top path) to 19800 (1200 cost and 1000 makespan reduction).
When considering the version of this scenario shown at the top right in Figure 4, the only change is that instead of one task with a duration of 2000, there are now 20 tasks with a duration of 100. Since tasks can be started on the next machine once they are finished, this will lead to a new makespan of 6100 (the top path will finish at 4200).

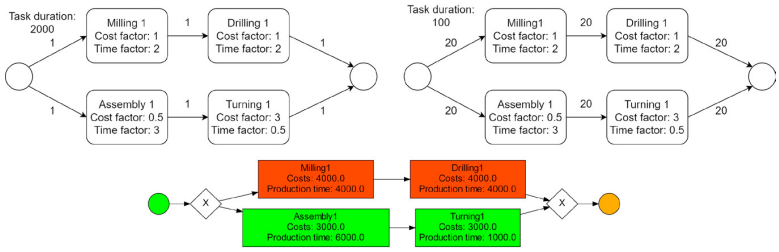


Fig. 4. Scenario 2 (Cost: color, Production time: width of the nodes)

When allowing the algorithm to place a cobot, the cobot now gets placed at the assembly workstation (high costs and the only makespan relevant workstation). Reducing the objective function from 20100 (14000 costs and a makespan of 6100) to 17370 (13100 costs and a makespan of 4370). This scenario demonstrates how the task duration can influence the cobot placement.

3.4. Scenario 3 - workstation groups

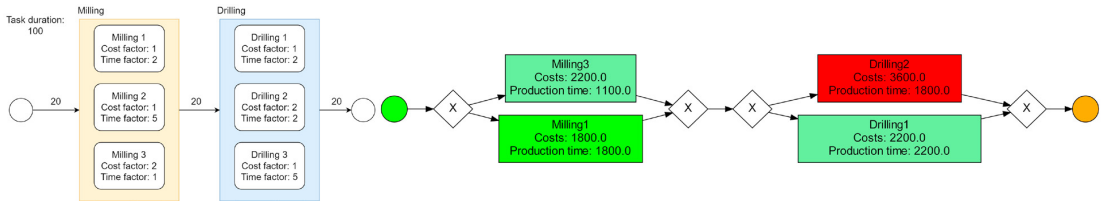


Fig. 5. Scenario 3 (Cost: color, Production time: width of the nodes)

In Figure 5 on the left hand side, a scenario with workstation groups can be seen. For each workstation group, an exclusive choice and a simple merge pattern is used. This means that for all 20 tasks a milling task has to be done before a drilling task. A milling task can be produced on all milling workstations and a drilling task can be done on all drilling workstations.
After running the genetic algorithm with 0 cobots, the BPMN model on the right side of Figure 5 can be mined from the log files. What can be seen is, that the two workstations with the highest time factor are not utilized at all. The drilling workstation with cost factor two gets only nine tasks assigned and still has the highest cost.
In Figure 6, the computed solution when the algorithm is allowed to place one cobot can be seen. Interestingly, the

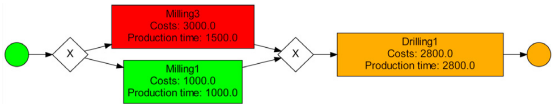


Fig. 6. Scenario 3 - 1 cobot (Cost = color, Production time = width of the nodes)

best solution is to place the cobot on the cheapest drilling workstation and then produce all tasks on this workstation, while the more expensive workstations are idle.

3.5. Scenario 4 - loop

In Figure 7, a rather simple scenario with three workstations is shown. Ten orders need a preparation, drilling, and packaging task. When allowing the placement of one cobot, this cobot gets placed on the preparation workstation,

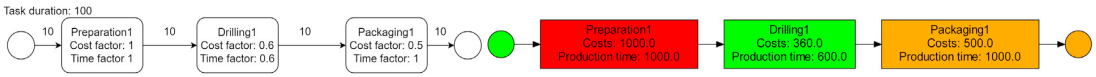


Fig. 7. Scenario 4.1 (Cost: color, Production time: width of the nodes)

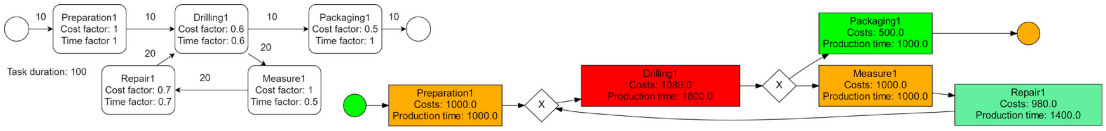


Fig. 8. Scenario 4.2 (Cost: color, Production time: width of the nodes)

because this workstation has the highest costs and the longest production time. When looking at Figure 8, it can be seen that this is an extended version of scenario 4.1. In this scenario, the drilling workstation has an error and an additional measure, repair, and drilling step is necessary two times, for each drilling task in scenario 4.1. This will result in the workflow pattern, a loop.
In this example, the drilling workstation has the lowest sum of production speed and cost factor. However, in the mined model on the right side of Figure 8, it is apparent that this workstation is part of the loop and is therefore highly utilized. Through this utilization, the workstation still has high costs and a long production time. When allowing the algorithm to place one cobot, this cobot gets deployed to the drilling in comparison to the preparation workstation of scenario 4.1.

3.6. Findings

Based on the discussed scenarios, we group properties that influence the placement of a cobot in two categories. The first category is properties that have a direct impact on the objective function:

- Time factor
- Cost factor
- Task amount
- Base duration of the tasks

These four properties result in production cost and production time and all four have a high impact on the placement of cobots.
The second category is properties of the scenario that can influence the placements of cobots, even if they do not directly influence the objective function:

- Workstation groups
- Scenario layout

Workstation groups allow the algorithm to shift tasks between workstations resulting in interesting choices as it is demonstrated in Figure 5. The layout of the scenario can also influence the cobot deployment in a meaningful way. An example would be the loop in Figure 8 that will favor assigning the cobot to a fast and cheap workstation.
The scenarios have been constructed to demonstrate what factors influence the placement of a cobot and show that process mining has huge potential to visualize and analyze the results of genetic algorithms. The default output for these algorithm runs would be a text file that shows the cobot-to-workstation assignment and the task to workstation assignment. For small scenarios, this could manually be transformed to visualizations as shown in this paper. However, when the scenarios increase in complexity, this is not possible any longer.

4. Conclusion and outlook

This paper demonstrates that process mining can interact with genetic algorithms in a way that visualizes the results of the genetic algorithm. Based on these visualizations, important properties for the placement of a cobot in a job shop scheduling problem are identified.
It is shown that implicit knowledge that is generated by utilizing a genetic algorithm or another heuristic solution approach on an extended cobot assignment and job shop scheduling problem can be analyzed with modern process mining approaches. The findings of this process mining, especially the cobot placement relevant properties, should help to enable prescriptive analytics for cobot placement in real-world companies.
State-of-the-art process mining might be applicable to analyze the optimization results from other optimization techniques and problems. This can allow companies to analyze and visualize optimization results where it was previously not possible. These visualizations can be used to gather implicit process knowledge as demonstrated in this paper and to improve the processes based on the generated knowledge. Process mining might be an interesting tool to better understand the results of optimization algorithms for any kind of real-world company or researcher.
In our upcoming work, the found factors that influence the placement of a cobot should be used in addition to implicit knowledge that is mined from event logs. This combination should be used to improve the quality of the results that are generated by a genetic algorithm or another metaheuristic on a large real-world data set.
Additionally, scenarios with conditional decisions and parallel workflow patterns should be reviewed. Such a conditional decision would be that, depending on a predecessor task, only a subgroup of successor tasks is allowed. For instance, that only successor workstations within a specific range to a workstation are allowed. This will mean, that orders can choose between different paths through the shop floor. In such scenarios, the physical position of a workstation will have an impact on the placement of cobots.

References

[1] Acampora, G., Vitiello, A., Di Stefano, B., van der Aalst, W., Günther, C., Verbeek, E., 2017. Ieee 1849tm: The xes standard. IEEE Computational Intelligence Magazine , 4–8.

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

[3] Bauer, W., Bender, M., Braun, M., Rally, P., Scholtz, O., 2016. Lightweight robots in manual assembly – best to start simply! Examining companies' initial experiences with lightweight robots. Technical Report. Fraunhofer Institute for Industrial Engineering IAO.

[4] Gall, M., Rinderle-Ma, S., 2020. Assessing process attribute visualization and interaction approaches based on a controlled experiment. Int. J. Cooperative Inf. Syst. 29, 2050007:1–2050007:33. https://doi.org/10.1142/S0218843020500070.

[5] Jämsä-Jounela, S.L., 2007. Future trends in process automation. Annual Reviews in Control 31, 211–220. URL: https://www.sciencedirect.com/science/article/pii/S1367578807000387, doi:https://doi.org/10.1016/j.arcontrol.2007.08.003. https://doi.org/10.1016/j.arcontrol.2007.08.003.

[6] Kinast, A., Doerner, K.F., Rinderle-Ma, S., 2021. Biased random-key genetic algorithm for cobot assignment in an assembly/disassembly job shop scheduling problem. Procedia Computer Science 180, 328–337. https://doi.org/10.1016/j.procs.2021.01.170.

[7] Leemans, S.J., Fahland, D., van der Aalst, W.M., 2013. Discovering block-structured process models from event logs-a constructive approach, in: International conference on applications and theory of Petri nets and concurrency, Springer. pp. 311–329.

[8] Lepenioti, K., Bousdekis, A., Apostolou, D., Mentzas, G., 2020. Prescriptive analytics: Literature review and research challenges. International Journal of Information Management 50, 57–70. https://doi.org/10.1016/j.ijinfomgt.2019.04.003.

[9] Mangat, A.S., Mangler, J., Rinderle-Ma, S., 2021. Interactive process automation based on lightweight object detection in manufacturing processes. Comput. Ind. 130, 103482. https://doi.org/10.1016/j.compind.2021.103482.

[10] Reinkemeyer, L., 2020. Process Mining in Action – Principles, Use Cases and Outlook. Springer International Publishing. https://doi.org/10.1007/978-3-030-40172-6.

[11] Rinderle-Ma, S., Mangler, J., 2021. Process automation and process miningin manufacturing, in: Int'l Conf. Business Process Management. (to appear).

[12] Russell, N., Ter Hofstede, A.H., Van Der Aalst, W.M., Mulyar, N., 2006. Workflow control-flow patterns: A revised view. BPM Center Report BPM-06-22, BPMcenter. org 2006.

[13] Siek, M., Mukti, R.M.G., 2020. Process mining with applications to automotive industry. IOP Conference Series: Materials Science and Engineering 924, 012033. URL: https://doi.org/10.1088/1757-899x/924/1/012033. https://doi.org/10.1088/1757-899x/924/1/012033.

[14] Stertz, F., Mangler, J., Scheibel, B., Rinderle-Ma, S., 2021. Expectations vs. experiences – process mining in small and medium sized manufacturing companies, in: Business Process Management (Forum). (accepted for publication).

[15] Van Der Aalst, W., 2016. Process Mining - Data Science in Action, Second Edition. Springer. https://doi.org/10.1007/978-3-662-49851-4.
[16] Vojić, S., 2020. Applications of collaborative industrial robots. Machines. Technologies. Materials. 14, 96–99.
[17] Wagner, S., Kronberger, G., Beham, A., Kommenda, M., Scheibenpflug, A., Pitzer, E., Vonolfen, S., Kofler, M., Winkler, S., Dorfer, V., Affenzeller, M., 2014. Architecture and design of the heuristiclab optimization environment, in: Advanced Methods and Applications in Computational Intelligence. Springer, pp. 197–261. https://doi.org/10.1007/978-3-319-01436-4_10.
[18] Website of Fraunhofer FIT - pm4py, . Fraunhofer fit - pm4py. https://pm4py.fit.fraunhofer.de/. Accessed: 2021-07-23.
[19] Website of the RISC Software GmbH, . Risc software gmbh. https://risc-software.at/. Accessed: 2021-07-23.
[20] Weckenborg, C., Kieckhäfer, K., Müller, C., Grunewald, M., Spengler, T.S., 2020. Balancing of assembly lines with collaborative robots. Business Research 13, 93–132. URL: https://link.springer.com/article/10.1007/s40685-019-0101-y. https://doi.org/10.1007/s40685-019-0101-y.

6 Optimizing the Solution Quality of Metaheuristics through Process Mining based on Selected Problems from Operations Research

Full publication details

Alexander Kinast, Roland Braune, Karl F. Doerner, and Stefanie Rinderle-Ma. "Optimizing the Solution Quality of Metaheuristics through Process Mining based on Selected Problems from Operations Research". Springer. 2023.

Author contributions

I am the main author of this paper. Therefore, I am responsible for the following:

- Data curation
  Maintain research data
- Conceptualization
  Formulation of research goals and aims
- Conceptualization
  Development of methodology
- Software
  Implementation of computer code and algorithms
- Writing - original draft & review
  Writing the original draft, review based on feedback and presentation at the conference
- Visualization
  Visualization of generated data

The co-authors are Roland Braune (software, formal analysis, validation, writing - draft & review), Karl F. Doerner (methodology, supervision, writing - review), Stefanie Rinderle-Ma (methodology, supervision, writing - draft & review).

Information on the Status
Submitted: 22.03.2023
Accepted: 24.05.2023
Published: Will be published on Springer (BPM Forum 2023)

Optimizing the Solution Quality of Metaheuristics through Process Mining based on Selected Problems from Operations Research

Alexander Kinast[1][0000−0002−0361−1136], Roland Braune[2][0000−0003−4086−229X], Karl F. Doerner[2][0000−0001−8350−1393], and Stefanie Rinderle-Ma[3][0000−0001−5656−6108]

[1] University of Vienna, Forschungsverbund Data Science, Kolingasse 14-16, 1090 Wien, Austria alexander.kinast@univie.ac.at
[2] University of Vienna, Department of Business Decisions and Analytics, Oskar-Morgenstern-Platz 1, 1090 Wien, Austria
[3] Technical University of Munich; TUM School of Computation, Information, and Technology, Boltzmannstrasse 3, 85748 Garching, Germany, stefanie.rinderle-ma@tum.de

Abstract. Methods from Operations Research (OR) are employed to address a diverse set of Business Process Management (BPM) problems such as determining optimum resource allocation for process tasks. However, it has not been comprehensively investigated how BPM methods can be used for solving OR problems, although process mining, for example, provides powerful analytical instruments. Hence, in this work, we show how process discovery, a subclass of process mining, can generate problem knowledge to optimize the solutions of metaheuristics to solve a novel OR problem, i.e., the combined cobot assignment and job shop scheduling problem. This problem is relevant as cobots can cooperate with humans without the need for a safe zone and currently significantly impact transitions in production environments. In detail, we propose two process discovery based neighborhood operators, namely process discovery change and process discovery dictionary change, and implement and evaluate them in comparison with random and greedy operations based on a real-world data set. The approach is also applied to another OR problem for generalizability reasons. The combined OR and process discovery approach shows promising results, especially for larger problem instances.

Keywords: Process Discovery · Operations Research · Metaheuristics · Memetic algorithm · Industry 4.0

1 Introduction

The application of techniques from Operations Research (OR) has been identified as promising *"avenue to obtain better processes"*, although *"OR techniques have not been systematically applied to solve process improvement problems*

yet" [2]. One example of the application of OR techniques to BPM is the allocation of resources to process tasks, e.g., [10]. Another example is the use of a memetic algorithm (MA) to mine change propagation behavior in process collaborations under confidential information, i.e., details on private processes [9]. Less attention has been paid to the reverse direction, i.e., how BPM methods such as process discovery (PD) can contribute to solve OR problems, even though PD provides powerful analytical instruments. [23] uses conformance checking to improve a scheduling problem in a hospital setting. In [13], we propose to use PD for visualization and exploration of solutions for a combined cobot assignment and job shop scheduling problem. In this case, the solutions that are generated by an MA are represented as process event logs. The discovered process models are then enriched by attributes such as cost and time for visual inspection and comparison of the solutions. In this work, we study how to exploit PD techniques for generating knowledge to optimize metaheuristics solutions based on two selected OR problems from the production domain, i.e., the cobot assignment and job shop scheduling problem [12] and flexible job shop scheduling problem [6] with an extended cobot assignment. As both problems are NP-hard optimization problems [29], metaheuristics offer promising solutions that are highly relevant in industry. MAs are one kind of metaheuristics that have proven useful for solving the cobot assignment and job shop scheduling problem [12]: a genetic algorithm explores the search space, and for promising solutions, a variable neighbourhood search (VNS) is performed. To investigate the potential of PD to metaheuristics, in this paper, we investigate i) how PD can be used in order to generate problem knowledge to optimize the solutions of the MA and ii) how much the solution quality can be increased. For this, we propose two PD-based neighbourhood operators, namely process discovery change and process discovery dictionary change. Both operators are implemented and evaluated alongside two standard neighbourhood operations, i.e., basic change and greedy change, based on three data sets for the two problems described above. The results underpin the potential of PD-based neighbourhood operations, especially for large data sets and many cobots.

Figure 1 depicts an overview of the overall idea and algorithm. On the left side in the operations research part of Fig. 1, it can be seen that an optimization problem is loaded and initial solutions for the problem are generated. After loading the problem, the main loop of the MA (detailed description in Sections 3 and 4) starts, and all individuals are evaluated. Whenever a new best solution is found, this solution is stored and a log file of this solution is created. In the BPM part of the Fig. 1, a local process model (LPM) is mined out of this log file and an LPM dictionary (described in Sections 4.3 and 4.4) is created. Knowledge generated with these models can now be used to boost the performance of the MA. The paper has the following outline. Section 2 discusses related work. Section 3 explains fundamental concepts for the work, such as memetic algorithm and local process models. In Section 4, the solution for the selected OR problems, including the PD-based neighbourhood operators, is described. Section 5 presents the

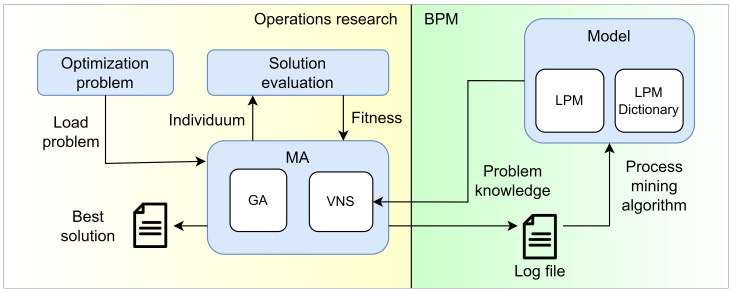


Fig. 1. MA with process discovery knowledge

computational experiments to show the performance of the algorithm. Section 6 concludes the paper.

2 Related Work

Scheduling problems in production are one of the hardest and most studied NP-hard optimization problems [29]. In [7], it is described that in the last decades, a lot of research has been done on developing efficient heuristic optimization algorithms for the (flexible) job shop scheduling problem due to its relevance for the industry. Especially local search methods like tabu search [25] have proven successful in this area. By combining the exploitation capabilities of local search with the explorative power of genetic algorithms [4], so-called MAs represent a hybrid between these two search paradigms. One of the most recent effective applications of such an algorithm to job shop scheduling is described in [30]. For process (re-)design, different OR methods have been used, e.g., mathematical programming [15]. [17] provides an overview of questions and approaches for automated planning in process design. OR methods are also used to determine the optimal data flow in process choreographies [14]. [5] put process model optimization to runtime based on formulating and solving a declarative process model plus temporal constraint as constraint satisfaction problem. Stochastic Petri nets [21,16] can be employed to model, simulate, and analyze dynamic process settings. PD has been mainly used to visualize and explore the results of the OR method to a given problem, so far. [8] uses process mining to analyze logs before scheduling in a hospital environment. In [13], we suggest using process mining to visually explore the results of cobot assignments by translating the schedules into logs. In [9], process mining is used to visualize and compare the solutions of an MA to predict change propagation behaviour in distributed process settings with and without confidentiality requirements. In [23] processes from a real-world clinic are improved: existing logs are analyzed, and a schedule is created with OR methods which is used to analyze the cause of deviations and

to improve the process. However, the aforementioned approaches do not exploit process mining techniques to optimize OR methods.

3 Fundamentals

This section presents background information on memetic algorithms and local process model mining as the two fundamental concepts combined in the solution method presented in this work.

3.1 Memetic Algorithms (MA)

In order to understand how MAs work, we introduce the fundamentals of genetic algorithms (GA) and local search (LS) methods. GAs are an abstraction of biological evolution. A set of solutions (population) is the basis. Selection, crossover, and mutation operations transform this initial set of solutions to the next generation. A selection operator selects two parents for the next generation. The idea is that fitter individuals are selected more often. The crossover operator now combines these two individuals and therefore mimics biological reproduction. The mutation operator can slightly change the produced offspring, similar to a natural mutation. By representing a problem as an individual of the population and creating a fitness function, that can assign fitness to new individuals, this basic genetic algorithm can solve a broad range of optimization problems [18]
LS methods start with a single solution. A set of local changes are applied to the starting solution, which will improve the starting solution until a local optimum has been found. Basic local search methods will stop once a local optimum has been found. However, there are algorithms that can escape local optima and continue the search. One of these algorithms is a VNS. This algorithm explores increasing neighbourhoods (a k^{th} neighbouring solution can be reached with k changes to the base solution), e.g. neighbourhoods with 1, 2, or 3 changes to the base solution. If an improvement to the best solution has been found, the algorithm is restarted from the newfound solution. [19]
A GA has a population and explores large parts of the search space. These GAs can be combined with LS so that the LS is applied to promising solutions that the GA finds. This MA combines global and local search methods and was able to provide good results for many practical problems. [20]

3.2 Local process model mining

Process models allow to *specify, describe, understand, and document processes more effectively than they can do using text*. Process models can be used to understand processes and make decisions [11]. Due to high concurrency and complex dependencies, simple sequence mining techniques do not work well on modern processes. However, process discovery (PD) algorithms have proven to

capture processes adequately based on event logs [3]. Local process model (LPM) mining is a PD technique introduced in [24] that aims at extracting the best LPM from an event log. LPMs are generated based on an initial set of process trees containing only one node, i.e., one workstation. These trees are assigned a fitness value based on five quality criteria, such as the number of traces that can be considered an instance of the LPM (support) and the harmonic mean of all explainable event occurrences divided by not explainable event occurrences (confidence) [24]. All or a subset of the process trees are selected for the next generation based on their fitness. The process trees are then expanded with different operators and nodes. This is necessary, as one node might be no good presentation for a large process. In the expansion step, a leaf is replaced by an operator. The original leaf is the first child of the new operator, and a second random node is the second child of the operator. This expansion step is done multiple times, until a stopping criterium is reached, and the best process tree is stored. These generated process trees can be converted into LPMs at any time.

4 Solution Method

This section describes the OR problems and the MA that has been used to solve them (see [12]). Moreover, this section defines the novel PD-based neighbourhood operators.

4.1 Operations Research Problems

We present the necessary details of the two OR problems based on which we investigate the potential of employing PD in metaheuristics. [12] describes orders and tasks. However, for clarity, orders and tasks will be called jobs and operations. [6] describes machines. However, for clarity, machines will be called workstations, as human workers can interact with machines and cobots on these workstations.

OR Problem 1 (Cobot assignment and job shop scheduling problem [12])
In this problem, a list of jobs is given. Each job contains multiple operations that are subject to precedence constraints. These operations should be executed on a given set of workstations.
All workstations that can do similar operations are grouped, e.g. all drilling workstations. Each workstation has a speed and cost factor. An example would be a new drilling workstation. This new workstation might have more expensive drills, but it is also faster than an old one. Furthermore, a predefined number of cobots can be deployed to workstations in order to speed up production as introduced in [28]. For each operation, a base cost and duration are available. Additionally, a workstation group (e.g. a drilling operation can be done on any drilling workstation) as well as precedence relations are given.
The objective function of this problem is a combination of normalized production cost and normalized makespan. An extension to the classical job-shop-scheduling

problem is that operations can be split and assigned to different workstations for quicker processing. Additionally, operation fragments can be executed in an arbitrary order.

OR Problem 2 (Flexible job shop scheduling [6]) *In this problem, a list of jobs is given. Each job contains multiple operations that are subject to precedence relations. Each operation can be processed on one out of a set of eligible workstations, and the processing time depends on the selected workstation.*
This base problem is extended by a cobot-to-workstation assignment.
The objective function of this problem is to minimize the makespan to finish all jobs. Therefore, operations must be assigned to the workstations, and a production order must be defined. The main difference to the first defined problem is the flexibility of operations (producible on many workstations instead of small workstation groups and the objective function).

Problems 1 + 2 are NP-hard problems extended by an additional decision aspect, i.e., a cobot-to-workstation-assignment. In [12], an MA has already been used to solve the cobot assignment and job shop scheduling problem. In this paper, we extend the MA with PD neighbourhood operators, i.e., if the genetic algorithm finds a promising solution, the VNS is started from this solution.

4.2 Encoding and evaluation

In Fig. 2, the encoding for Problems 1 + 2 is shown. The first part of the encoding is the operation to workstation assignment. If an operation can be produced on multiple workstations, the upper bound equals the number of these workstations. The value represents which of the possible workstations is used for production. E.g. if workstations 0 to 4 are possible for production, the upper bound is 4, and a value of two would mean that the second workstation is used.
In the second part of the encoding in Fig. 2, each operation's priority is encoded. If multiple operations can be produced simultaneously at the same workstation, the operation with the highest priority gets produced first. Each number between zero and the largest possible integer is possible. E.g. two tasks, task 1 with priority 5 and task 2 with priority 10, should be produced on the same workstation. The task with the highest priority, namely task 2, is produced first.
The final part of the encoding is the cobot-to-workstations-assignment that can be seen on the right side of Fig. 2. The upper bound of this value is the number of workstations that have no cobot assigned. The value represents which of these workstations a cobot should be assigned. E.g. if workstations 0 to 10 have no cobot assigned, the upper bound of the value is 10 and a value of 5 would mean that a cobot is assigned to workstation 5.
In [28], it is described that a cobot speeds up production by 30%. This value is used for the evaluation.

Details regarding the evaluation of the extended cobot assignment and job shop scheduling problem can be found in [12]. During the evaluation of one solution, two objective values (production cost and makespan) are generated.

The objective function F that is used as fitness is a combination of normalized cost and normalized makespan, i.e., $F = n_{cost} + n_{makespan}$.

Instances of the extended cobot assignment and flexible job shop scheduling problem do not define production costs. Therefore, the makespan should be minimized in these instances.

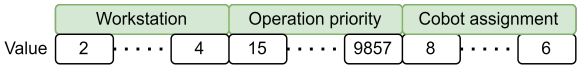


Fig. 2. Integer-based encoding

4.3 Local process model mining

In this work, an LPM represents highly used workstations and relations between these workstations in the currently best solution of the algorithm. More precisely, the best-rated LPM is mined whenever the MA finds a new best solution. Section 4.4 will explain how one or multiple LPMs are used inside the MA to improve the algorithm's performance. However, the conceptual idea is that solutions that are close to the best-found solution might improve if more operations are assigned to those highly used workstations of the best solution.
Compared to [24], the computational effort is crucial in the context of this paper. Hence, the LPM mining algorithm is adjusted to be executable in the MA. For this, the number of operators to build the LPMs is limited to sequence and xor operators. Regarding the described problems, this deviation from the originally proposed mining algorithm does not have disadvantages, as the problems are defined without loops and concurrency. Additionally, generating all possible solutions in the selection step is not feasible. Therefore, a random subset is generated before the expansion step. Figure 3 shows the process of generating an

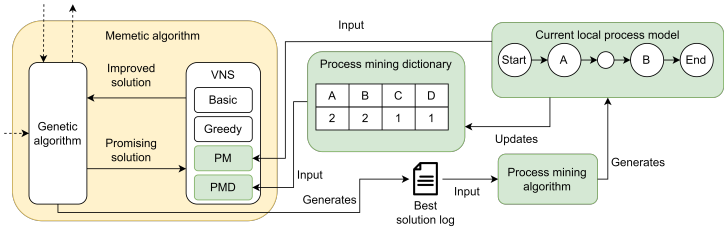


Fig. 3. Generation and usage of local process models

LPM during the run of the MA. Green parts have been developed or adjusted

for this paper. The basis for the LPM is the event log representing the current best solution found by the MA.

During the evaluation of one solution, information regarding jobs, order of operations, workstations, and cobot placement is available. The generation of an event log in the xes format [1] is triggered every time a new best solution is found. The xes file contains a trace for each job. The traces, in turn, contain events for each operation. Each event defines a start and end timestamp (start and end of the operation), a job ID, an operation ID, which workstation has been used and the information if a cobot is currently assigned to this workstation. Using the adjusted LPM mining algorithm, the best LPM for this given log file is created. An example would be the sequential execution on workstations A and B in a problem with four workstations A-D. Section 4.4 will explain how LPMs are used in the VNS.

4.4 Memetic algorithm

To generate neighbourhood solutions in a MA as described in Section 3, a neighbourhood operator applies k changes to an initial solution.
Independent of the neighbourhood operator, each part of the encoding, described in Fig. 2, has an equal chance of being selected for change (operation-to-workstation assignment, operation priority, cobot assignment). The first neighbourhood operator is the **Basic change (B)**. In this change, one value of the selected part is randomized within its bounds. The second operator is the **Greedy change (G)**. Regarding operator priority and cobot assignment, this change equals the basic change. In the operation-to-workstation assignment, all workstations that have a cobot assigned are calculated. Workstations with cobots have a threefold probability of being selected during the operation-to-workstation assignment. The third operator is the **Process discovery change (PD)**. Regarding operator priority and cobot assignment, this change equals the basic change. In the operation-to-workstation assignment, the latest LPM is used. This can be seen in Fig. 3. Workstations that are part of the latest LPM have a threefold probability of being selected compared to other workstations. The final operator is the **Process discovery dictionary change (PDD)**. Regarding operator priority, this change is equal to the basic change. In the operation-to-workstation assignment and the cobot-to-workstation assignment, the weight of each workstation is the weight of the entry in the process mining dictionary. This can be seen in Fig. 3. This means that workstations that greatly impact the process over multiple generations of LPMs have a higher chance of being selected.

In Algorithm 1, the evaluation of the MA (cf. [12]) with PD-based VNS is described. In line 0, a solution and one neighbourhood operator are passed to the evaluation method. A fitness value for this solution, called solutionFitness, is generated. This can be seen in Algorithm 1 in line 1. In line 2, it is checked if the VNS should be applied. It is applied to solutions within a given range of the best solution that has been found so far. Lines 3, 4, 5, 14, and 15 indicate the minimum number of individuals generated whenever the VNS is started. An example would be k_{max}=5, where at least 50 solutions with $k \in \{1, 3, 5\}$ changes

Algorithm 1. Pseudo code - MA with process discovery

Parameters/Methods	Description
BestSolution	Best solution found so far
BestSolutionLog	Log file describing the best solution found so far
BestFitness	Best fitness found so far
VnsThreshhold	Threshold to check if the VNS should be applied
LPM	Local process model of the best solution found so far
LPMDictionary	Dictionary that is updated based on mined LPMs
UpdateDictionary()	Method that updated the current LPMDictionary
EvaluateSolution()	Method to get the quality of a passed individual
SolutionLog()	Method to get the log file of a solution
MineLocalProcessModel()	Method to mine a LPM out of a process log
NeighbouringSolution()	Method that generates a solution with k changes

```
0  Evaluate(solution, neighbourhood, k_max)
1      solutionFitness ← EvaluateSolution(solution)
2      if(solutionFitness ≤ BestFitness * VnsThreshhold)
3          k ← 1
4          while(k ≤ k_max)
5              for(i = 0, i ≤ 50, i++)
6                  newSolution ← NeighbouringSolution(solution, neighbourhood, k)
7                  newSolutionFitness ← EvaluateSolution(s')
8                  if(newSolutionFitness < solutionFitness)
9                      solutionFitness ← newSolutionFitness
10                     solution ← newSolution
11                     k ← 1
12                     goto line 3
13                 end if
14             end for
15             k += 2
16         end while
17     end if
18     if(solutionFitness < BestFitness)
19         BestFitness = solutionFitness
20         BestSolution = solution
21         BestSolutionLog = SolutionLog(solution)
22         LPM = MineLocalProcessModel(BestSolutionLog)
23         LPMDictionary = UpdateDictionary(LocalProcessModel)
24     end if
25     return x
```

are generated. In line 6, k changes are made to the existing best solution based on the passed neighbourhood operator of line 0. Four neighbourhood operators, i.e., basic change, greedy change, PD change, and PDD change, are used in line 6 and will be explained in detail after the algorithm description. In line 7, the fitness of the new changed solution is evaluated.

The VNS is restarted on a first-improvement basis. This can be seen in lines 8 to 13. The found improved solution replaces the current best solution for this

variable neighbourhood.
Lines 18 to 24 show that the MA also stores the best-found solution. If a new best solution is found and the neighbourhood operator is one of the PD operators, a solution log of this solution is generated and a LPM mining algorithm is applied. The mined LPM replaces the LPM of the currently best solution. It is assumed that key workstations of existing solutions are part of the LPM. Examples are workstations that are used frequently in the best existing solution. Utilizing this information in the genetic algorithm might be good for already promising solutions to assign operations to these workstations.
Fig. 3 shows the development of a PDD. A dictionary with all workstations is created to utilize information extracted from multiple LPMs. Each workstation has a base weight of 1. Each time a new LPM is mined, the weight of all workstations that are part of this LPM is increased by one.

5 Numerical experiments

5.1 Data and code

The problem files for the cobot assignment and job shop scheduling problem can be found at https://doi.org/10.5281/zenodo.7691316 and the problem files for the cobot assignment and flexible job shop scheduling problem at https://doi.org/10.5281/zenodo.7691455. Algorithm 1 is implemented in C# and embedded into HeuristicLab, a framework for heuristic optimization [26]. The simulation framework Easy4Sim[4] was used to evaluate solutions. The code is provided at https://zenodo.org/badge/latestdoi/614876607.
The evaluation of the approach necessitates large-scale computational experiments. For this, the HPC3 cluster[5] in Vienna was used. All calculations were executed on nodes with a Xeon-G 6226R CPU 2.9 GHz. To execute the C# code on a Linux cluster, the mono framework [27] was utilized. To evaluate the overhead of the runtime environment, preliminary experiments were conducted. Stretching the computation time by a factor of 1.6 allows for a similar number of solutions to be evaluated compared to the same code running on a native .NET platform (MS Windows). All runs of the MA have been done on the HPC. Therefore, this factor has been used for all runs of the MA, and the original runtime is reported in this paper.

5.2 Constraint programming formulation

A constraint programming (CP) formulation for all solved problems has been done to measure the implemented algorithm's performance. If the CP model terminates, it finds the global optimum of a problem. Therefore the CP model gives an overview of the complexity of the problem (can the optimum be found in a reasonable time?). If no optimum is found, it gives a good base quality which

[4] https://www.risc-software.at/
[5] https://w3.vdc.univie.ac.at/wiki/index.php/Slurm

can be compared to the solutions found by the developed algorithm.
The exact formulation for the first two data sets can be found in [12]. To solve the third problem, minor adjustments have been made to the CP model. Since the CP formulation of the problem requires a lot of space, it is not presented here. IBM ILOG CP Optimizer has been employed for implementing the model and for solving the example problems. The model definition can be found at https://doi.org/10.5281/zenodo.7754794.

5.3 Data dimensions

Three different data sets were solved with all neighbourhood operators. In [12], the first two data sets are explained. The first data set is a combined cobot assignment and job shop scheduling problem from the industry. It has 54 workstations, 210 jobs, and 1265 operations. This instance is split into two halves and four quarters to create additional smaller instances. The second data set is inspired by this real-world data set and has 50 artificial instances. These instances are similar in size compared to real-world instances (full, halves, quarters). In [6], the third data set is introduced. This data set contains large flexible job shop scheduling instances that are extended with a cobot-to-workstation-assignment in this paper. The instance size ranges from 30×10 (jobs $\times$ workstations) to 100×20.

5.4 Real-world cobot assignment and job shop scheduling problem

The real-world problem described in [12] has been solved with the following parameters:

- Runtime: Short (100 minutes), medium (200 minutes), and long (300 minutes) runtime
- Cobots: 0, 5, and 10
- neighbourhood operator: Basic change, greedy change, PD change, PDD change
- Instances: Full, halves, quarters
- Repetitions: 10

These settings result in 2520 runs of the MA. The reported runtime is used for the full instance and 30%, and 10% of this runtime is used for the half and quarter instances, respectively. In [12], the CP model has been used to solve the real-world data set with zero and five cobots.

In Table 2, the average normalized objective value of all runs of the MA is reported and compared to the CP results. Both values of the objective function (makespan, cost) are normalized so that a higher normalized value represents a better value. The maximum of each normalized value is 1, which means the closer the objective value gets to 2, the better the result is. The values in the cells represent the average for 10, 20, and 40 runs of the algorithm for the full instance, the half instances, and the four quarters, respectively.

Table 2. Detailed results for the real-world problem

		0 Cobots			5 Cobots			10 Cobots		
NBOp	Time	Quarters	Halves	Full	Quarters	Halves	Full	Quarters	Halves	Full
B	Short	1.2698	0.9654	1.0644	1.7878	1.5955	1.6491	1.7926	1.5995	1.6414
G		1.2213	0.9574	1.0423	1.7565	1.5714	1.6583	1.7884	1.6245	1.7343
PD		1.2475	0.8933	1.0262	1.7604	1.5790	1.6714	1.8168	1.6277	1.7423
PDD		1.2766	0.9653	1.0722	1.6111	1.5351	1.5504	1.8176	1.6200	1.7279
B	Medium	1.2876	0.9915	1.0854	1.8227	1.6115	1.7227	1.8489	1.6563	1.7329
G		1.2886	0.9845	1.0646	1.8074	1.6018	1.7164	1.8343	1.6516	1.7457
PD		1.2838	1.0436	1.0821	1.8134	1.5967	1.7071	1.8431	1.6746	1.7511
PDD		1.2877	1.0349	1.1154	1.6463	1.6624	1.7268	1.8540	1.6776	1.7919
B	Long	1.3004	1.0897	1.1099	1.8322	1.6244	1.7506	1.8584	1.7019	1.7617
G		1.2968	1.0587	1.0685	1.8215	1.6738	1.7394	1.8455	1.6829	1.7461
PD		1.2946	1.0322	1.0973	1.8257	1.6738	1.7237	1.8616	1.6936	1.7975
PDD		1.2954	1.0520	1.0930	1.6660	1.6756	1.7814	1.8672	1.6921	1.8086
CP		0.9057	1.1493	1.0216	-2.4404	-1.0479	0.9258			

NBOp: neighbourhood Operator; B: Basic; G: Greedy; PD(D): Process discovery (Dictionary)

The colored cells mark the best neighbourhood for each combination of runtime, instances size, and the number of cobots. This highlights the advantages of the different neighbourhood operators.
The PD operators try to identify important workstations in generated solutions. The PDD operator even learns over a large number of generations. Since applying PD operators comes with an overhead, the instance must be hard enough that this generated knowledge has enough impact in the remaining time. In Table 2, it can be seen that the PD operators, especially the PDD operator, outperform other neighbourhood operators on complex problems (large instance, high number of cobots) if enough runtime is given.
Once the number of cobots increases, the CP model has difficulties finding a valid solution. This can be seen in the last line of Table 2. The CP approach delivers good zero cobot results, especially for the half instances. However, with five cobots, the CP formulation already has trouble finding valid solutions.

5.5 Generated cobot assignment and job shop scheduling problem

The second data set solved is the artificial data set described in [12]. In this data set, 50 instances in 3 sizes have been created:

- Small instances (10 instances)
  300 operations / 30 workstations
- Medium instances (20 instances)
  600 operations / 30 workstations
  600 operations / 50 workstations
- Large instances (20 instances)
  1200 operations / 30 workstations
  1200 operations / 50 workstations

All instances have been solved with the following parameters:

- Cobots: 0, 5, and 10

– neighbourhood operator: Basic change, greedy change,
  PD change, PDD change
– Repetitions: 10

These settings result in 6000 runs of the algorithm. The runtime was 60, 180, and 300 minutes for the small, medium, and large instances, respectively.

Table 3. Detailed results for the artificially generated instances

	0 cobots			5 cobots			10 cobots		
neighbourhood	Small	Medium	Large	Small	Medium	Large	Small	Medium	Large
B	0.8140	0.5258	0.3657	0.9704	0.6342	0.5450	1.1592	0.7708	0.6781
G	0.8124	0.4991	0.3483	0.9463	0.6188	0.5399	1.1331	0.7570	0.6792
PD	0.8192	0.5243	0.3883	0.9767	0.6473	0.5576	1.1462	0.7799	0.6801
PDD	0.8123	0.5355	0.3897	1.0100	0.6632	0.5426	1.1903	0.7953	0.6815
CP	0.4139	0.2989	0.1240	0.4898	0.2989	-0.2926	0.5690	0.2989	-0.2926

NBOp: neighbourhood Operator; B: Basic; G: Greedy; PD(D): Process discovery (Dictionary)

Table 3 summarizes the performance of the neighbourhood operators compared to the CP model on the artificial instances. The value in each cell represents the normalized objective value (normalized cost + normalized makespan) with an upper bound of 2. A larger value means that on average better solutions have been found. The coloured cells represent the best neighbourhood operator with regard to the instance size and the number of cobots.
If the CP model did not find a solution for a cobot setting, the solution with fewer cobots is taken. It can be seen that the MA outperforms the CP model for this problem. This is independent of the used neighbourhood.
Table 3 shows that the PD neighbourhood operators outperform the basic and greedy neighbourhood operators over the whole data set. This is again especially true for the PDD operator. Which performs, on average, 2.4% better than the basic neighbourhood, 4.5% better than the greedy neighbourhood, and 1.5% better than the PD neighbourhood.
The values in the table represent the average over 100 and 200 instances for the small and medium/large instances, respectively. Due to the larger number of instances, results from this data set are less prone to errors than the real-world instances.

5.6 Cobot assignment and flexible job shop scheduling problem

The third data set solved is the flexible job shop scheduling problem described in [6], cf. Problem 2. For the previous two problems, it could be seen that the CP results have performed better for simpler instances and worse for complex instances. In this problem, CP delivers good results due to the simple, makespan-only objective. To compete with the CP formulation with an equal runtime, minor adaptations had to be done in the MA.
A fraction of the initial population of the MA has been initialized with solutions generated using priority dispatching rules. These priority rules allow the generation of acceptable initial solutions that can be further improved with the MA.

A considerable amount of priority rules are described in [22]. The following have been used in the operation-to-workstation assignment in this paper:

- Most work remaining
- Shortest processing time remaining
- Most operations remaining
- Operational flow slack per processing time
- Flow due date per work remaining
- Shortest processing time per work remaining
- Shortest processing time and work in next queue

Additionally, the *Highest number of assignable operations* and the *Largest amount of assignable work* priority rules have been designed for the cobot-to-workstation assignment. In the first cobot-to-workstation rule, workstations are sorted by the number of operations that can be assigned and the available cobots are assigned to the top workstations. In the second rule, the sorting is done by the duration of all assignable operations on a specific workstation.
Additionally, generating LPMs has been stopped until the first generation is finished. Four problem files (0, 10, 20, 30) of two categories (smallest and largest) were selected. The smallest category has 30 jobs with 10 workstations, and the largest has 100 jobs with 20 workstations. These problems were solved with cobots assigned to 0%, 20%, and 40% of the workstations. Each problem was solved with all four described neighbourhood operators and 20 repetitions. This resulted in 1920 runs of the algorithm. The CP solver and the MA had a runtime limit of 60 minutes.

Table 4. Detailed makespan results for the cobot assignment and FJSP

	B	**G**	**PD**	**PDD**	**CP**
small 0%	764	764	765	764	762
small 20%	702	703	703	702	699
small 40%	650	650	650	649	646
large 0%	3906	3906	3906	3906	3904
large 20%	3587	3587	3587	3587	3587
large 40%	3314	3314	3314	3314	3317

Table 4 reports the average solution quality for the MA and the CP model of the flexible job shop scheduling instances. The values represent the average objective value (makespan) across each instance group. Hence, smaller values indicate a better solution quality. The CP solver delivered good results for all numbers of cobots for the small instances. With growing problem difficulty (increasing number of cobots and instance size), the performance of the MA increased.
For the large instances with 40% cobots, a slight advantage of the MA over the CP model can be observed. Even though the performance of the neighbourhood

operators is pretty similar, the PDD operator outperforms the other neighbourhood operators again and delivers the best results for the most complex (largest, highest amount of cobots) instances.

6 Summary and Outlook

This paper introduces a novel combination of an MA with a feedback loop that utilizes LPM mining. This MA uses different neighbourhoods that utilize the information generated with this PD algorithm, and the results are compared to traditional neighbourhood operators and a CP model.
Two problems from OR were tackled to show the algorithm's generalizability. The algorithm should be easily adaptable to new problems due to the flexibility of the base algorithm, the genetic algorithm. Additionally, it has been implemented in HeuristicLab, which can, due to its plugin-based architecture, easily be extended with new problem formulations.
Running additional code like the PD algorithms during the execution of a genetic algorithm to generate knowledge comes with overhead. This knowledge can help identify important parts of the process.
A series of experiments on different problems were started to quantify the impact of this generated knowledge. This paper reports the results of 10440 runs of the MA. A CP formulation was employed for all problems to have a baseline performance measure.
For small instances and simple problems, the overhead incurred through PD inhibits the competitiveness of our approach. However, it was shown that neighbourhood operators that utilize PD algorithms to generate knowledge outperform other neighbourhood operators and the CP model on large and complex instances. In further research, different metaheuristics, feedback variants, problems, and PD algorithms can be reviewed. In the current version, the order and connections between workstations in the LPM is not utilized, however, utilizing this information might be helpful in upcoming research.

Acknowledgments

This work has been partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 277991500. Additionally, we are thankful that the RISC Software GmbH allowed us to use the simulation framework, Easy4Sim.

References

1. Ieee standard for extensible event stream (xes) for achieving interoperability in event logs and event streams. IEEE Std 1849-2016 pp. 1–50 (2016). https://doi.org/10.1109/IEEESTD.2016.7740858

2. van der Aalst, W.M.P., Rosa, M.L., Santoro, F.M.: Business process management - don't forget to improve the process! Bus. Inf. Syst. Eng. **58**(1), 1–6 (2016). https://doi.org/10.1007/s12599-015-0409-x
3. van der Aalst, W.: Process discovery: Capturing the invisible. IEEE Computational Intelligence Magazine **5**(1), 28–41 (2010). https://doi.org/10.1109/MCI.2009.935307
4. Affenzeller, M., Wagner, S., Winkler, S., Beham, A.: Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications. Chapman and Hall/CRC, 1 edn. (2009). https://doi.org/10.1201/9781420011326
5. Barba, I., Jiménez-Ramírez, A., Reichert, M., Valle, C.D., Weber, B.: Flexible runtime support of business processes under rolling planning horizons. Expert Syst. Appl. **177**, 114857 (2021). https://doi.org/10.1016/j.eswa.2021.114857
6. Braune, R., Benda, F., Doerner, K.F., Hartl, R.F.: A genetic programming learning approach to generate dispatching rules for flexible shop scheduling problems. International Journal of Production Economics **243**, 108342 (2022). https://doi.org/https://doi.org/10.1016/j.ijpe.2021.108342
7. Chaudhry, I.A., Khan, A.A.: A research survey: review of flexible job shop scheduling techniques. International Transactions in Operational Research **23**(3), 551–591 (2016). https://doi.org/https://doi.org/10.1111/itor.12199
8. Cunzolo, M.D., Guastalla, A., Aringhieri, R., Sulis, E., Amantea, I.A., Ronzani, M., Francescomarino, C.D., Ghidini, C., Fonio, P., Grosso, M.: Combining process mining and optimization: A scheduling application in healthcare. In: Business Process Management Workshops. pp. 197–209 (2022). https://doi.org/10.1007/978-3-031-25383-6_15
9. Fdhila, W., Rinderle-Ma, S., Indiono, C.: Memetic algorithms for mining change logs in process choreographies. In: Service-Oriented Computing. pp. 47–62 (2014). https://doi.org/10.1007/978-3-662-45391-9_4
10. Ihde, S., Pufahl, L., Völker, M., Goel, A., Weske, M.: A framework for modeling and executing task-specific resource allocations in business processes. Computing **104**(11), 2405–2429 (2022). https://doi.org/10.1007/s00607-022-01093-2
11. Kesari, M., Chang, S., Seddon, P.B.: A content-analytic study of the advantages and disadvantages of process modelling. ACIS 2003 Proceedings p. 2 (2003)
12. Kinast, A., Braune, R., Doerner, K.F., Rinderle-Ma, S., Weckenborg, C.: A hybrid metaheuristic solution approach for the cobot assignment and job shop scheduling problem. Journal of Industrial Information Integration **28**, 100350 (2022). https://doi.org/https://doi.org/10.1016/j.jii.2022.100350
13. Kinast, A., Doerner, K.F., Rinderle-Ma, S.: Combining metaheuristics and process mining: Improving cobot placement in a combined cobot assignment and job shop scheduling problem. Procedia Computer Science **200** (2022). https://doi.org/https://doi.org/10.1016/j.procs.2022.01.384
14. Köpke, J., Franceschetti, M., Eder, J.: Optimizing data-flow implementations for inter-organizational processes. Distributed Parallel Databases **37**(4), 651–695 (2019). https://doi.org/10.1007/s10619-018-7251-3
15. Kumar, A., Liu, R.: Business workflow optimization through process model redesign. IEEE Trans. Engineering Management **69**(6), 3068–3084 (2022). https://doi.org/10.1109/TEM.2020.3028040
16. Leemans, S.J.J., Maggi, F.M., Montali, M.: Reasoning on labelled petri nets and their dynamics in a stochastic setting. In: Business Process Management. pp. 324–342 (2022). https://doi.org/10.1007/978-3-031-16103-2_22

17. Marrella, A., Chakraborti, T.: Applications of automated planning for business process management. In: Business Process Management. pp. 30–36 (2021). https://doi.org/10.1007/978-3-030-85469-0_4
18. Mitchell, M.: Genetic algorithms: An overview. In: Complex. vol. 1, pp. 31–39. Citeseer (1995)
19. Mladenović, N., Hansen, P.: Variable neighborhood search. Computers & Operations Research **24**(11), 1097–1100 (1997). https://doi.org/https://doi.org/10.1016/S0305-0548(97)00031-2, https://www.sciencedirect.com/science/article/pii/S0305054897000312
20. Moscato, P., Cotta, C.: An Accelerated Introduction to Memetic Algorithms, pp. 275–309. Springer International (09 2018). https://doi.org/10.1007/978-3-319-91086-4_9
21. Rogge-Solti, A., Weske, M.: Prediction of business process durations using non-markovian stochastic petri nets. Inf. Syst. **54**, 1–14 (2015). https://doi.org/10.1016/j.is.2015.04.004
22. Sels, V., Gheysen, N., Vanhoucke, M.: A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions. International Journal of Production Research **50**, 4255–4270 (08 2012). https://doi.org/10.1080/00207543.2011.611539
23. Senderovich, A., Weidlich, M., Yedidsion, L., Gal, A., Mandelbaum, A., Kadish, S., Bunnell, C.A.: Conformance checking and performance improvement in scheduled processes: A queueing-network perspective. Inf. Syst. **62**, 185–206 (2016). https://doi.org/10.1016/j.is.2016.01.002
24. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.: Mining local process models. Journal of Innovation in Digital Ecosystems **3**(2), 183–196 (2016). https://doi.org/https://doi.org/10.1016/j.jides.2016.11.001
25. Vilcot, G., Billaut, J.C.: A tabu search algorithm for solving a multicriteria flexible job shop scheduling problem. International Journal of Production Research **49**(23), 6963–6980 (2011). https://doi.org/https://doi.org/10.1080/00207543.2010.526016
26. Wagner, S., Kronberger, G., Beham, A., Kommenda, M., Scheibenpflug, A., Pitzer, E., Vonolfen, S., Kofler, M., Winkler, S., Dorfer, V., Affenzeller, M.: Architecture and Design of the HeuristicLab Optimization Environment, pp. 197–261. Springer (2014). https://doi.org/10.1007/978-3-319-01436-4_10
27. Mono framework. https://www.mono-project.com/, accessed: 2023-02-15
28. Weckenborg, C., Kieckhäfer, K., Müller, C., Grunewald, M., Spengler, T.S.: Balancing of assembly lines with collaborative robots. Business Research **13**(1), 93–132 (2020). https://doi.org/https://doi.org/10.1007/s40685-019-0101-y
29. Xie, J., Gao, L., Peng, K., Li, X., Li, H.: Review on flexible job shop scheduling. IET Collaborative Intelligent Manufacturing **1**(3), 67–77 (2019). https://doi.org/https://doi.org/10.1049/iet-cim.2018.0009
30. Zhang, G., Sun, J., Lu, X., Zhang, H.: An improved memetic algorithm for the flexible job shop scheduling problem with transportation times. Measurement and Control **53**(7-8), 1518–1528 (2020). https://doi.org/https://doi.org/10.1177/0020294020948094

7 Conclusion

7.1 Results

All results that are presented in this thesis have the grand goal of combining metaheuristics and process mining to improve the performance of metaheuristics on state-of-the-art scheduling processes. The main results of this thesis are:

1. In [55], it is described that using cobots will give many businesses a competitive edge.
   Scheduling operative tasks to different operational resources, such as human workers and cobots, introduce a new problem. The complexity of this problem increases with the number of cobots that should be considered. [56]
   In Chapters 3.4, 4.4, and 6.2, encodings for a set of different extended cobot assignment and scheduling problems are developed, which are:

   - Extended cobot assignment and job shop scheduling problem
   - Extended cobot assignment and assembly line balancing problem
   - Extended cobot assignment and flexible job shop scheduling problem

   These encodings are the foundation that can be used for a broad range of metaheuristics to generate solutions for these problems.

2. *The development of algorithms for dealing with data will be one of major challenges in Industry 4.0* [57]
   In this thesis, the current state-of-the-art algorithms additions to these algorithms are used for large numerical studies. These studies generate solutions for existing and newly introduced extended cobot assignment and scheduling problems. The algorithms used are:

   - A genetic algorithm (Chapter 3.4-3.6)
   - A memetic algorithm (Chapter 4.4-4.5)
   - A knowledge-based memetic algorithm that utilizes process mining methods (Chapter 6.2-6.3)

   To analyze the performance of the individual algorithms, the results are compared to results that already exist in the literature (Chapter 4.5). A CP model is created for instances where no solution existed in the literature (Chapters 4.3 and 6.3), and the results of the algorithms are compared to the results of the CP model. The CP model is able to find the global optimum for simple instances and can give a baseline for other instances.

3. Process mining can extract knowledge from event logs and links data mining, business process modelling and analysis. [58]

In Chapter 6.2, a novel knowledge-based memetic algorithm utilizes this knowledge extraction from event logs to generate local process models as described in [47]. These local process models contain knowledge that is used in the variable neighbourhood search of this knowledge-based memetic algorithm. Extracting and utilizing this knowledge comes with an overhead. However, it is shown in Chapter 6.3, that this algorithm outperforms traditional approaches on complex extended cobot assignment and scheduling problems.

4. The slowest throughput workstation sets the flow rate of assembly lines. This bottleneck workstation might change depending on different reasons. Adding a flexible resource such as a cobot to a bottleneck workstation can allow large savings. [59]
   In Chapter 4.5, a numerical study on a real-world problem instance on the impact of different numbers of cobots is done. It is shown that the first cobot significantly impacts the objective function. However, this rapidly declines with increasing amounts of cobots. With this algorithm, the ideal amount of cobots for a specific production system can be calculated.

5. Genetic algorithms can be easily adapted to new problems. However, it is often not clear how the results of the genetic algorithm can be interpreted and if found solutions are efficient. [60]
   Key features of process mining are the visualization of business processes and the analysis of processes [61].
   Chapter 5.3 in this work presents how process mining algorithms can be used to visualize the results of a genetic algorithm and highlight different solution attributes with colours and sizes. Various instances are created, and solutions are analyzed to get an insight into a typical black box solution.

6. *Being able to build on existing knowledge is key to scientific progress.* [62]
   To allow the reproducibility of results shown in this thesis, the code and all problem definitions used to generate the results in Chapter 6, are shared in Chapter 6.3.

7.2 Discussion

In Chapter 1.5, the following major research questions have been defined:

A) What is the best way to plan resources such as cooperative robots in deterministic manufacturing processes?

B) Can process mining methods generate additional insights into manufacturing processes?

C) Can process mining algorithms improve the performance of metaheuristics?

All defined minor and major research questions have been answered during this work and a detailed list of contributions is given in Chapter 1.6. In this discussion, a really short overview about the solved research questions is given, and which areas of the work are not fully explored and might be promising for future research.
To answer **RQ A**, a set of new encodings have been developed to allow the encoding of flexible resources. To answer **RQ B**, process mining algorithms have been used on metaheuristic solutions to generate additional insight into manufacturing processes. These solutions have been visualized.
Utilizing process mining to visualize metaheuristic results has helped understanding the underlying process. Trying different process mining methods and visualization techniques might be helpful for real-world black box optimizations.
Feedback possibilities have been explored in **RQ C**, and generated knowledge from **RQ B** has been used in a feedback loop to boost the performance of a memetic algorithm.
The feedback loop in the memetic algorithm looked promising, and it might be worth exploring different feedback strategies and process mining methods in the future.

7.3 Outlook

Industrial production environments currently experience a large shift towards industry 4.0 technologies such as Internet of Things, big data, 3D printing, cloud computing, and cobots. These changes will drastically increase the data collected in production environments, as current technologies allow us to measure and log more data than ever.
It is important that companies can estimate the impact of such new technologies on already existing production environments. Therefore, problem formulations must be adapted to reflect the current needs of the industry. This is shown for three different problems in this thesis. For each of these problems, a problem encoding has been developed.
Additionally, new algorithms need to be developed to benefit from these large quantities of data. Process mining is a research area that greatly benefits from these production environment changes. Due to strong requests from the industry, many new process mining algorithms and frameworks have been developed. These allow generating different process models for previous black box processes. These process models can be the basis for an in-depth process analysis that can give additional insight into the process, which was impossible before.
Over the last decades, metaheuristics have been successfully applied to nearly all kinds of problems that occur in industries. However, due to the black-box nature of metaheuristics, it is often impossible to learn from generated solutions. This work has shown that a combination of metaheuristics and process mining can give additional insight into manufacturing processes, which allows to boost the performance of traditional metaheuristics.

Since both research areas, process mining and metaheuristics, individually play an important role in the industrial transformation towards Industry 4.0, it can be expected that the number of available algorithms will increase. In this work, one specific interaction between these two research areas has been demonstrated. Extracting knowledge from good metaheuristic solutions and utilising that knowledge in the variable neighbourhood search of a memetic algorithm. As this combination looks promising for complex industrial problems, other combinations might help to get away from classical black-box optimisation towards knowledge-based algorithms.

References

[1] C. Santos, A. Mehrsai, A. Barros, M. Araújo, and E. Ares, "Towards industry 4.0: an overview of european strategic roadmaps," *Procedia manufacturing*, vol. 13, pp. 972–979, 2017. https://doi.org/10.1016/j.promfg.2017.09.093.

[2] O. Bongomin, G. Gilibrays Ocen, E. Oyondi Nganyi, A. Musinguzi, and T. Omara, "Exponential disruptive technologies and the required skills of industry 4.0," *Journal of Engineering*, vol. 2020, pp. 1–17, 2020. https://doi.org/10.1155/2020/4280156.

[3] J. Xu, E. Huang, L. Hsieh, L. H. Lee, Q.-S. Jia, and C.-H. Chen, "Simulation optimization in the era of industrial 4.0 and the industrial internet," *Journal of Simulation*, vol. 10, no. 4, pp. 310–320, 2016. 10.1057/s41273-016-0037-6.

[4] F. Sherwani, M. M. Asad, and B. S. K. K. Ibrahim, "Collaborative robots and industrial revolution 4.0 (ir 4.0)," in *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*, pp. 1–5, IEEE, 2020. https://doi.org/10.1109/ICETST49965.2020.9080724.

[5] A. Adriaensen, F. Costantino, G. Di Gravio, and R. Patriarca, "Teaming with industrial cobots: A socio-technical perspective on safety analysis," *Human Factors and Ergonomics in Manufacturing & Service Industries*, vol. 32, no. 2, pp. 173–198, 2022. https://doi.org/10.1002/hfm.20939.

[6] A. Pauliková, Z. Gyurák Babel'ová, and M. Ubárová, "Analysis of the impact of human–cobot collaborative manufacturing implementation on the occupational health and safety and the quality requirements," *International Journal of Environmental Research and Public Health*, vol. 18, no. 4, p. 1927, 2021. https://doi.org/10.3390/ijerph18041927.

[7] Z. M. Bi, C. Luo, Z. Miao, B. Zhang, W. Zhang, and L. Wang, "Safety assurance mechanisms of collaborative robotic systems in manufacturing," *Robotics and Computer-Integrated Manufacturing*, vol. 67, p. 102022, 2021. https://doi.org/10.1016/j.rcim.2020.102022.

[8] C. Weckenborg, K. Kieckhäfer, C. Müller, M. Grunewald, and T. S. Spengler, "Balancing of assembly lines with collaborative robots," *Business Research*, vol. 13, pp. 93–132, 2020. http://dx.doi.org/10.1007/s40685-019-0101-y.

[9] A. R. Sadik and B. Urban, "Flow shop scheduling problem and solution in cooperative robotics—case-study: One cobot in cooperation with one worker," *Future Internet*, vol. 9, no. 3, p. 48, 2017. http://dx.doi.org/10.3390/fi9030048.

[10] T. Yamada and R. Nakano, "Job shop scheduling," *IEE control Engineering series*, pp. 134–134, 1997.

[11] S. M. Almufti, "Historical survey on metaheuristics algorithms," *International Journal of Scientific World*, vol. 7, no. 1, p. 1, 2019. http://dx.doi.org/10.14419/ijsw.v7i1.29497.

[12] J. H. Holland, "Genetic algorithms and the optimal allocation of trials," *SIAM journal on computing*, vol. 2, no. 2, pp. 88–105, 1973. https://doi.org/10.1137/0202009.

[13] T. Bartz-Beielstein, J. Branke, J. Mehnen, and O. Mersmann, "Evolutionary algorithms," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, no. 3, pp. 178–195, 2014. http://dx.doi.org/10.1002/widm.1124.

[14] D. Bertsimas and J. Tsitsiklis, "Simulated annealing," *Statistical science*, vol. 8, no. 1, pp. 10–15, 1993. 10.1214/ss/1177011077.

[15] M. Kumar, D. Husain, N. Upreti, D. Gupta, *et al.*, "Genetic algorithm: Review and application," *Available at SSRN 3529843*, 2010. https://dx.doi.org/10.2139/ssrn.3529843.

[16] S. Tsutsui, A. Ghosh, D. Corne, and Y. Fujimoto, "A real coded genetic algorithm with an explorer and an exploiter populations.," in *ICGA*, pp. 238–245, 1997.

[17] F. Neri and C. Cotta, "Memetic algorithms and memetic computing optimization: A literature review," *Swarm and Evolutionary Computation*, vol. 2, pp. 1–14, 2012. https://doi.org/10.1016/j.swevo.2011.11.003.

[18] M. Halaška and R. Šperka, "Process mining – the enhancement of elements industry 4.0," in *2018 4th International Conference on Computer and Information Sciences (ICCOINS)*, pp. 1–6, 2018. 10.1109/ICCOINS.2018.8510578.

[19] W. Van Der Aalst, A. Adriansyah, A. K. A. De Medeiros, F. Arcieri, T. Baier, T. Blickle, J. C. Bose, P. Van Den Brand, R. Brandtjen, J. Buijs, *et al.*, "Process mining manifesto," in *Business Process Management Workshops: BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I 9*, pp. 169–194, Springer, 2012. http://dx.doi.org/10.1007/978-3-642-28108-2.

[20] O. Dogan and O. F. Gurcan, "Data perspective of lean six sigma in industry 4.0 era: A guide to improve quality," in *Proceedings of the International Conference on Industrial Engineering and Operations Management*, pp. 943–953, IEOM Society Southfield, MI, USA, 2018.

[21] L. Reinkemeyer, “Status and future of process mining: from process discovery to process execution,” in *Process Mining Handbook*, pp. 405–415, Springer, 2022. https://doi.org/10.1007/978-3-031-08848-3_13.

[22] M. Jans, J. M. van der Werf, N. Lybaert, and K. Vanhoof, “A business process mining application for internal transaction fraud mitigation,” *Expert Systems with Applications*, vol. 38, no. 10, pp. 13351–13359, 2011. https://doi.org/10.1016/j.eswa.2011.04.159.

[23] E. Mahendrawathi, N. Arsad, H. M. Astuti, R. P. Kusumawardani, and R. A. Utami, “Analysis of production planning in a global manufacturing company with process mining,” *Journal of Enterprise Information Management*, vol. 31, no. 2, pp. 317–337, 2018. http://dx.doi.org/10.1108/JEIM-01-2017-0003.

[24] E. Benmelech and M. Zator, “Robots and firm investment,” tech. rep., National Bureau of Economic Research, 2022. 10.3386/w29676.

[25] S. Chen, J. Montgomery, and A. Bolufé-Röhler, “Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution,” *Applied Intelligence*, vol. 42, pp. 514–526, 2015. 10.1007/s10489-014-0613-2.

[26] M. N. Omidvar, X. Li, and X. Yao, “A review of population-based metaheuristics for large-scale black-box global optimization—part i,” *IEEE Transactions on Evolutionary Computation*, vol. 26, no. 5, pp. 802–822, 2021. 10.1109/TEVC.2021.3130838.

[27] A. Muluk, H. Akpolat, and J. Xu, “Scheduling problems—an overview,” *Journal of Systems Science and Systems Engineering*, vol. 12, no. 4, pp. 481–492, 2003.

[28] J. Xie, L. Gao, K. Peng, X. Li, and H. Li, “Review on flexible job shop scheduling,” *IET Collaborative Intelligent Manufacturing*, vol. 1, no. 3, pp. 67–77, 2019. https://doi.org/10.1049/iet-cim.2018.0009.

[29] M. Drótos, G. Erdős, and T. Kis, “Computing lower and upper bounds for a large-scale industrial job shop scheduling problem,” *European Journal of Operational Research*, vol. 197, no. 1, pp. 296–306, 2009. https://doi.org/10.1016/j.ejor.2008.06.004.

[30] P. Winklehner and V. A. Hauder, “Flexible job-shop scheduling with release dates, deadlines and sequence dependent setup times: A real-world case,” *Procedia Computer Science*, vol. 200, pp. 1654–1663, 2022. https://doi.org/10.1016/j.procs.2022.01.366.

[31] Z. Tian, X. Jiang, W. Liu, and Z. Li, “Dynamic energy-efficient scheduling of multi-variety and small batch flexible job-shop: A case study for the aerospace industry,” *Computers & Industrial Engineering*, vol. 178, p. 109111, 2023. https://doi.org/10.1016/j.cie.2023.109111.

[32] M. Yousefi, M. Yousefi, D. Hooshyar, and J. A. de Souza Oliveira, “An evolutionary approach for solving the job shop scheduling problem in a service industry,” *International Journal of Advances in Intelligent Informatics*, vol. 1, no. 1, pp. 1–6, 2015. https://doi.org/10.26555/ijain.v1i1.5.

[33] P. Coelho, A. Pinto, S. Moniz, and C. Silva, “Thirty years of flexible job-shop scheduling: a bibliometric study,” *Procedia Computer Science*, vol. 180, pp. 787–796, 2021. https://doi.org/10.1016/j.procs.2021.01.32.

[34] A. E. Ezugwu, A. K. Shukla, R. Nath, A. A. Akinyelu, J. O. Agushaka, H. Chiroma, and P. K. Muhuri, “Metaheuristics: a comprehensive overview and classification along with bibliometric analysis,” *Artificial Intelligence Review*, vol. 54, pp. 4237–4316, 2021. https://doi.org/10.1007/s10462-020-09952-0.

[35] D. Wolpert and W. Macready, “No free lunch theorems for optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997. 10.1109/4235.585893.

[36] I. Fister Jr, X.-S. Yang, I. Fister, J. Brest, and D. Fister, “A brief review of nature-inspired algorithms for optimization,” *arXiv preprint arXiv:1307.4186*, 2013. https://doi.org/10.48550/arXiv.1307.4186.

[37] C. Blum, A. Roli, and E. Alba, “An introduction to metaheuristic techniques,” *Parallel Metaheuristics: A New Class of Algorithms*, vol. 47, p. 1, 2005. https://doi.org/10.1002/0471739383.ch1.

[38] M. Mitchell, “Genetic algorithms: An overview.,” in *Complex.*, vol. 1, pp. 31–39, Citeseer, 1995. https://doi.org/10.1002/cplx.6130010108.

[39] M. Dorigo and C. Blum, “Ant colony optimization theory: A survey,” *Theoretical Computer Science*, vol. 344, no. 2, pp. 243–278, 2005. https://doi.org/10.1016/j.tcs.2005.05.020.

[40] M. H. Fazel Zarandi, A. A. Sadat Asl, S. Sotudian, and O. Castillo, “A state of the art review of intelligent scheduling,” *Artificial Intelligence Review*, vol. 53, pp. 501–593, 2020. https://doi.org/10.1007/s10462-018-9667-6.

[41] W. van der Aalst, “Process discovery: Capturing the invisible,” *IEEE Computational Intelligence Magazine*, vol. 5, pp. 28–41, Feb 2010. 10.1109/MCI.2009.935307.

[42] F. A. Yasmin, F. A. Bukhsh, and P. D. A. Silva, “Process enhancement in process mining: a literature review,” in *CEUR workshop proceedings*, vol. 2270, pp. 65–72, Rheinisch Westfälische Technische Hochschule, 2018.

[43] S. J. Leemans, D. Fahland, and W. M. Van Der Aalst, "Process and deviation exploration with inductive visual miner.," *BPM (demos)*, vol. 1295, no. 8, 2014.

[44] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004. https://doi.org/10.1109/TKDE.2004.47.

[45] A. Weijters and J. Ribeiro, "Flexible heuristics miner (fhm)," in *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pp. 310–317, 2011. 10.1109/CIDM.2011.5949453.

[46] A. K. A. de Medeiros, A. J. Weijters, and W. M. van der Aalst, "Genetic process mining: an experimental evaluation," *Data mining and knowledge discovery*, vol. 14, pp. 245–304, 2007.

[47] N. Tax, N. Sidorova, R. Haakma, and W. M. van der Aalst, "Mining local process models," *Journal of Innovation in Digital Ecosystems*, vol. 3, no. 2, pp. 183–196, 2016. https://doi.org/10.1016/j.jides.2016.11.001.

[48] W. Song, S. Liu, and Q. Liu, "Business process mining based on simulated annealing," in *2008 The 9th International Conference for Young Computer Scientists*, pp. 725–730, 2008. 10.1109/ICYCS.2008.279.

[49] J. Buijs, B. van Dongen, and W. van der Aalst, "A genetic algorithm for discovering process trees," in *2012 IEEE Congress on Evolutionary Computation*, pp. 1–8, 2012. 10.1109/CEC.2012.6256458.

[50] S. M. Hejazi, M. Zandieh, and M. Mirmozaffari, "A multi-objective medical process mining model using event log and casual matrix," *Healthcare Analytics*, p. 100188, 2023. https://doi.org/10.1016/j.health.2023.100188.

[51] D. Viner, M. Stierle, and M. Matzner, "A process mining software comparison," 2021. https://doi.org/10.48550/arXiv.2007.14038.

[52] U. Nuşin, "Event data visualization through process mining: a case for emergency medical service system in adana," *European Journal of Technique (EJT)*, vol. 9, no. 2, pp. 320–329, 2019. https://doi.org/10.36222/ejt.575564.

[53] E. Ruschel, E. A. P. Santos, and E. d. F. R. Loures, "Establishment of maintenance inspection intervals: an application of process mining techniques in manufacturing," *Journal of Intelligent Manufacturing*, vol. 31, pp. 53–72, 2020. https://doi.org/10.1007/s10845-018-1434-7.

[54] M. A. Cuendet, R. Gatta, A. Wicky, C. L. Gerard, M. Dalla-Vale, E. Tavazzi, G. Michielin, J. Delyon, N. Ferahta, J. Cesbron, *et al.*, "A differential process mining analysis of covid-19 management for cancer patients,"

Frontiers in Oncology, vol. 12, p. 1043675, 2022. 10.3389/fonc.2022.1043675.

[55] M. Javaid, A. Haleem, R. P. Singh, S. Rab, and R. Suman, "Significant applications of cobots in the field of manufacturing," *Cognitive Robotics*, vol. 2, pp. 222–233, 2022. https://doi.org/10.1016/j.cogr.2022.10.001.

[56] A. R. Sadik, A. Taramov, and B. Urban, "Optimization of tasks scheduling in cooperative robotics manufacturing via johnson's algorithm case-study: One collaborative robot in cooperation with two workers," in *2017 IEEE Conference on Systems, Process and Control (ICSPC)*, pp. 36–41, 2017. 10.1109/SPC.2017.8313018.

[57] Y. Lu, "Industry 4.0: A survey on technologies, applications and open research issues," *Journal of Industrial Information Integration*, vol. 6, pp. 1–10, 2017.

[58] B. Hannewijk, "Process mining in manufacturing," 2021.

[59] Y. Cohen, S. Shoval, and M. Faccio, "Strategic view on cobot deployment in assembly 4.0 systems," *IFAC-PapersOnLine*, vol. 52, no. 13, pp. 1519–1524, 2019. 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019.

[60] E. Hart and P. Ross, "Gavel - a new tool for genetic algorithm visualization," *IEEE Transactions on Evolutionary Computation*, vol. 5, pp. 335–348, Aug 2001. 10.1109/4235.942528.

[61] P. Badakhshan, B. Wurm, T. Grisold, J. Geyer-Klingeberg, J. Mendling, and J. vom Brocke, "Creating business value with process mining," *The Journal of Strategic Information Systems*, vol. 31, no. 4, p. 101745, 2022. https://doi.org/10.1016/j.jsis.2022.101745.

[62] A. Laurinavichyute, H. Yadav, and S. Vasishth, "Share the code, not just the data: A case study of the reproducibility of articles published in the journal of memory and language under the open data policy," *Journal of Memory and Language*, vol. 125, p. 104332, 2022. https://doi.org/10.1016/j.jml.2022.104332.