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].

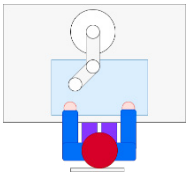

Fig. 1. Human-cobot interaction

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]

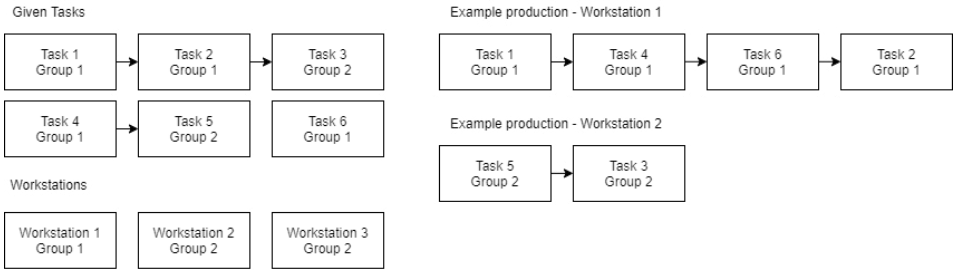


Fig. 2 Precedence graph

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].

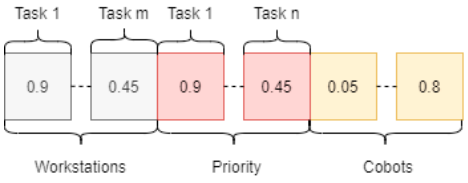


Fig. 3. Biased random-key encoding

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.

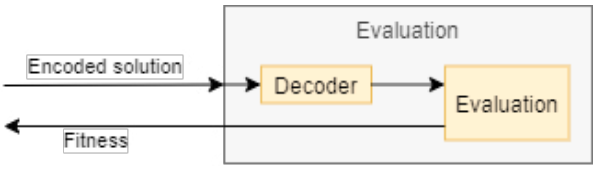


Fig. 4. Decoding and evaluation

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 |

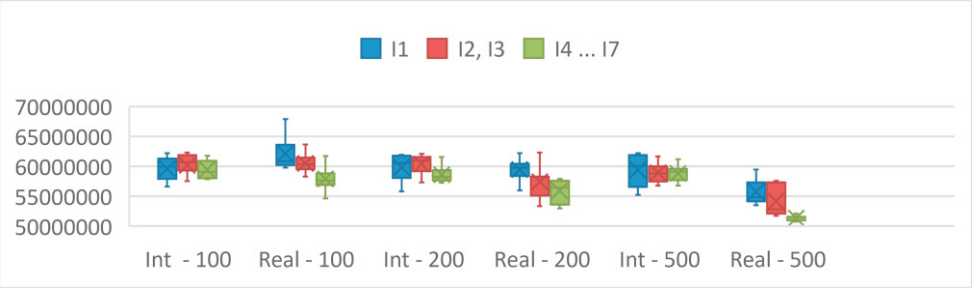


Fig. 5. Computational results with standard deviation

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.

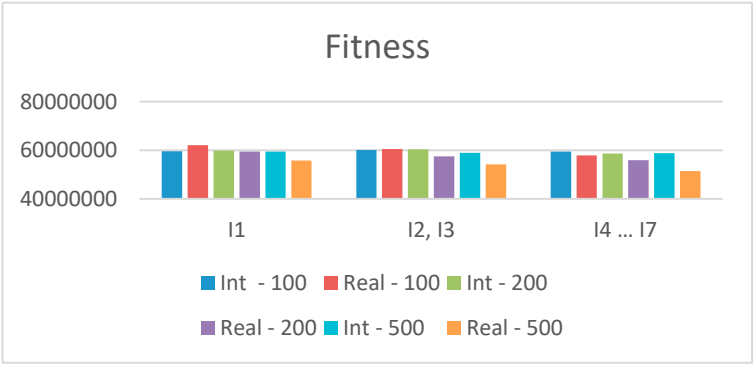


Fig. 6. Aggregated results

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.