

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

Alexander Kinast¹[0000-0002-0361-1136], Roland Braune²[0000-0003-4086-229X],
Karl F. Doerner²[0000-0001-8350-1393], and
Stefanie Rinderle-Ma³[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

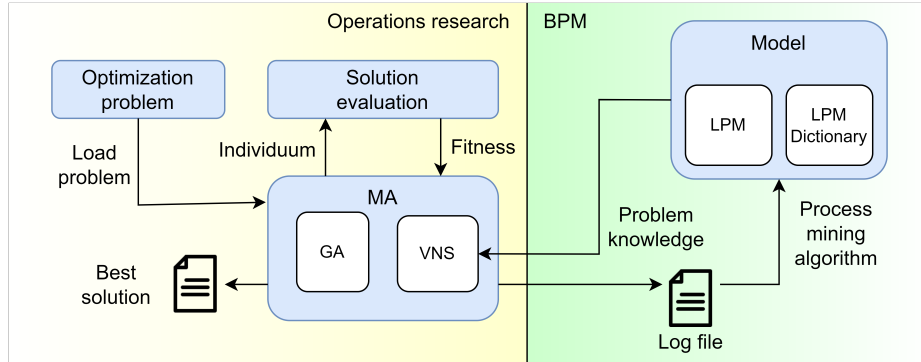


Fig. 1. MA with process discovery knowledge

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.

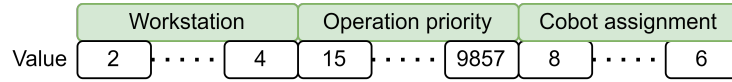


Fig. 2. Integer-based encoding

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

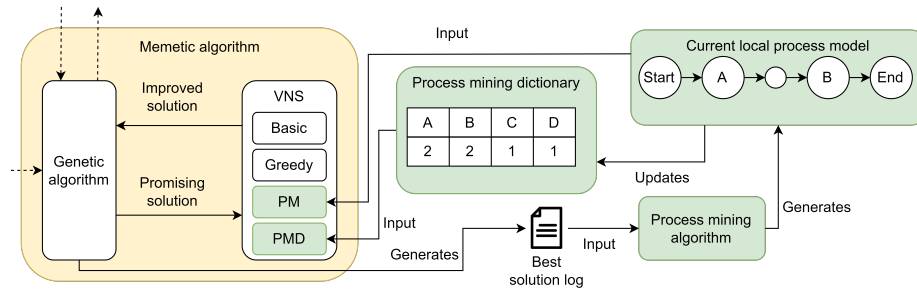


Fig. 3. Generation and usage of local process models

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
VnsThreshold	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, kmax)
1   solutionFitness ← EvaluateSolution(solution)
2   if(solutionFitness ≤ BestFitness * VnsThreshold)
3     k ← 1
4     while(k ≤ kmax)
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 Easy4Sim⁴ 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 cluster⁵ 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

NBOP	Time	0 Cobots			5 Cobots			10 Cobots		
		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

neighbourhood	0 cobots			5 cobots			10 cobots		
	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://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>