Computers & Industrial Engineering 174 (2022) 108760

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

Scheduling and batching with evolutionary algorithms in simulation–optimization of an industrial formulation plant☆

Christian Klanke *, Sebastian Engell

Process Dynamics and Operations Group, Department of Biochemical and Chemical Engineering, TU Dortmund University, Emil-Figge-Str. 70, 44227 Dortmund, Germany

ARTICLE INFO

Keywords:
Batching
Industrial batch process scheduling
Simulation–optimization
Evolutionary algorithm
Discrete-event simulation

ABSTRACT

Industrial scheduling problems are usually characterized by a high complexity and in many, if not all cases, a detailed representation of many specific features of the production process and of the constraints is necessary to ensure that the resulting schedules are feasible when they are executed. Such detailed representations are provided by commercial discrete-event simulation programs which are often employed in the planning phase to detect bottlenecks and to validate design decisions, but usually do not provide optimized schedules. In this paper, we combine a tailored Evolutionary Algorithm with a high-fidelity simulation model in a Simulation–Optimization approach in order to obtain optimized executable production schedules.

The industrial production process considered here is a formulation plant that consists of a formulation stage and a filling stage with optional intermediate storage in a set of buffer tanks. The operational constraints include limited operator availability and shift constraints, sequence-dependent changeover times in both stages, product- and line-dependent processing rates, and the need for synchronization of operations. The objective is to minimize the tardiness of a set of orders some of which have long operation times and small slack between the earliest possible end times and the due dates.

The orders consist of a certain number of batches, but if all the batches of an order are processed on the same equipment, meeting the due dates is not possible. Therefore the orders have to be decomposed into sub-orders (production orders) to realize parallel processing of urgent orders on several units with the goal to meet due dates. The decision to distribute the total demand of the individual customer orders to production orders is also called batching decision in the literature.

A distinctive feature of our approach is that three independent encodings of the batching, allocation and sequencing decisions are used. The batching decisions lead to variable-length chromosomes for the allocation and sequencing encodings. For the treatment of variable-length chromosomes, new crossover and mutation operators, as well as repair algorithms are presented.

The results for three test cases show that the introduction of the batching decisions significantly improves the tardiness of the production schedules. A thorough tuning of the categorical parameters of the Evolutionary Algorithm has been performed and it is shown that the resulting parameter setting yields reproducible results, while outperforming simpler approaches.

1. Introduction

Many processes from the process and manufacturing industry can be improved with respect to their economic performance or to their ecological footprint without the need for substantial investment into new processing equipment, by improving the resource utilization of the existing equipment. Optimal production scheduling is among the measures to achieve such improvements. However, realizing these improvements is a challenge in complex production environments. Algorithmic optimization frameworks are necessary to support human planners and schedulers in their decision making in order to fully realize the potential of the plants.

☆ Acknowledgements: This work was partially funded by the European Regional Development Fund (ERDF) in the context of the project OptiProd.NRW (https://www.optiprod.nrw/en). We thank Dominik Bleidorn, Engelbert Pasieka, Christian Koslowski, Vassilios Yfantis and Christian Sonntag for providing comments on the manuscript.

* Corresponding author.
E-mail addresses: christian.klanke@tu-dortmund.de (C. Klanke), sebastian.engell@tu-dortmund.de (S. Engell).

https://doi.org/10.1016/j.cie.2022.108760
Received 28 May 2022; Received in revised form 7 September 2022; Accepted 19 October 2022
Available online 25 October 2022

In demand-driven production the task that has to be solved in planning and scheduling is the satisfaction of a set of customer orders, each of which specifies a certain amount of a certain product and a delivery date. When the volumes of the orders and their requested production resources or their due dates differ largely and the orders can be processed on different units, splitting orders into production orders that can be processed on different units in parallel helps to meet the due dates. If this distribution of customer orders to production orders is not performed by an upper layer planning function, the task of deriving the production orders must be integrated into the scheduling solution (see Harjunkoski et al., 2014). In the process industries, the generation of production orders from customer orders is commonly referred to as batching. A production order can also consist of several batches. The batches of one production order are produced sequentially on the same piece of equipment, so the scheduling decisions are taken for production orders. The orders could instead be split completely into individual batches which are handled by the scheduling algorithm, but if the orders consist of many batches this increases the combinatorial complexity of the problem enormously and an efficient production requires to avoid changeovers between products as much as possible. Therefore the introduction of production orders that comprise a certain number of batches that are processed without changeovers in between makes sense.

From the point of view of optimal scheduling, batching decisions complicate the optimization problem significantly. The splitting of orders renders the number of orders and therefore the number of scheduling decisions variable and dependent on the specific batching decisions taken. How to solve the investigated scheduling problem with integrated batching decisions is therefore one of the main challenges that is tackled in this work.

Detailed models of the production processes are often necessary to ensure that the resulting schedules are feasible when they are executed on shop floor level. Frequently, such models are already available within a commercial simulation software and have been used to discover bottlenecks in the production processes or to make and analyse design decisions. In contrast, rigorous optimization formulations of scheduling problems are usually based upon simplified process representations to keep the size of the resulting mixed-integer optimization problem manageable and also because of the required modelling effort. Therefore a relevant approach to schedule optimization for industrial-scale problems is to combine such available high-fidelity simulation models with optimization methods. Specifically, in this contribution we integrate an industrial-strength discrete-event simulator with a tailored Evolutionary Algorithm (EA).

When it comes to the application of simulation–optimization in an industrial environment, there are different stakeholders involved in the deployment and operation of the scheduling solution. Two of these are the end-users and the solution providers. A sufficient fidelity of the model, a high solution quality as well as ease-of-use of the software, are in the interest of the end-users. On the other hand, extensibility and maintainability are main concerns for solution providers. One goal of this work is to develop an approach that satisfies the needs of both groups by proposing an implementation of Simulation Optimization (SO) where the optimizer and the simulator are only loosely coupled and where an industry-proven Discrete-Event Simulator (DES) for schedule generation is used. The EA which is used as the optimizer uses largely independent encodings of the different scheduling decisions and interfaces with the simulation as an external component. This reduces the effort for transferring the optimization approach to other industrial cases.

To the best of the authors' knowledge, batching decisions have not been investigated in the setting of a meta-heuristic optimization approach for an industrial case study where allocation and sequencing decisions are taken at the same time. Meta-heuristics are generally considered as not being able to solve the batching problem together with the allocation and sequencing problem at all (Harjunkoski et al., 2014, Table 6.1).

The remainder of this contribution is structured as follows: In Section 2 the relevant literature is discussed. In Section 3, the industrial formulation plant that is used as a case study is introduced. In Section 4, the details of the optimization approach, such as the encodings of the batching, allocation and sequencing decisions, the newly developed genetic operators, as well as the repair algorithms are presented. Then the results section follows where comparisons of the optimization results with different simpler approaches is made and the reproducibility of the solution quality is investigated. Our contribution is finalized with conclusions on the results of the approach and ideas for future work.

2. Literature review

In the process industries, a batch represents the physical or chemical transformation of defined amounts of substances and keeps its identity throughout the process, as it is not mixed or split and the final packaged products only consist of the material from a single batch. This differentiates the term batch as used in the process industries from its usage in the manufacturing industry where a batch represents a group of items or jobs (Voß & Witt, 2003). Batching refers to the splitting of orders into batches, possibly of variable size. Other terms for which the meaning overlaps with batching are lot sizing, campaigning, order-splitting and lot streaming. In Potts and van Wassenhove (1992) lot sizing denotes the decision about how many jobs to process at the same time on a unit that is capable of processing several jobs at once. A campaign usually denotes a relatively large slice of the total planning horizon that is dedicated to the repetitive and exclusive production of a single product (Fumero et al., 2012; Papageorgiou & Pantelides, 1993; Shah & Pantelides, 1991). Lot streaming is similar to what we denote by batching, but products of a sub-lot only proceed to the next stage if all batches of that sub-lot have been finished in the previous stage (Defersha & Chen, 2012). Order splitting is defined as the decision to split an order into several sub-orders that can be processed independently and simultaneously on parallel units (Kim & Su, 2020, Kim & Lee, 2021, Lee & Kim, 2021). This definition is indeed identical to our definition of batching. We decided to consistently use the term batching here, as it is more established in the Process Systems Engineering community.

After batching or order splitting, it must be decided, where, i.e. on which production units, and when to produce the production orders. In many cases, the timing of the operations can directly be derived from the sequence of operations because a zero-wait schedule cannot be improved by introducing waiting times. Then production scheduling comprises the degrees of freedom of batching, allocation and sequencing (Harjunkoski et al., 2014).

Research has brought forth a wide variety of methods to solve optimal production scheduling problems with these three degrees of freedom. Both exact and approximate, i.e. heuristic and meta-heuristic, methods have been developed. Among the exact approaches, Mixed-Integer Linear Programming (MILP) and Constraint Programming (CP) methods are the dominant ones (Méndez et al., 2006). Examples of successful applications to industrial-scale problems that include allocation and sequencing decisions have been described, among others, in Awad et al. (2022) or Escobet et al. (2019) for CP and in Basán et al. (2020), Georgiadis et al. (2019) or Klanke et al. (2020) for MILP.

CP and MILP-based scheduling formulations can include the batching degree of freedom. Due to the increase of the complexity of the problem, besides monolithic and approaches where all degrees of freedom are considered simultaneously, often a sequential approach is employed where the batching decisions are taken first and then optimal schedules are computed.

Examples for monolithic MILP- or CP-formulations that include batching, sequencing and allocation decisions, can be found in Sundaramoorthy and Maravelias (2008a, 2008b), Castro et al. (2008),

Prasad and Maravelias (2008), Lee and Maravelias (2017), Ackermann et al. (2022) (MILP) and Novas (2019) (CP).

A demonstration of a sequential approach is provided in Schwindt and Trautmann (2000). They solved the problem of determining the number of batches with a simple heuristic, after which the batch scheduling problem, i.e. the determination of allocation and sequence of the batches, is solved. An alternative approach, where an MILP model is solved to obtain the number (and sizes) of batches before the detailed scheduling is done in a subsequent step can be found in Roe et al. (2005). Further discussion of sequential batching and scheduling approaches can be found in Maravelias (2012).

For medium-sized problems, or problems with a limited number of features and constraints, exact methods are a reasonable choice. However, for the application of these methods to large-scale, industrial problems the modelling depth is insufficient in many cases.

As discussed in Honkomp et al. (2000) a major issue in the day-to-day usage of a schedule optimization framework is that schedules are dead on arrival. As a thought experiment, take the delivery of raw material: Since the detailed process that underlies the raw material delivery may be too involved to be modelled in a MILP or CP formulation, its effect is commonly lumped into the processing time of a batch. Then, as soon as the delivery does not meet the (average) delivery time that has been assumed, the solution may be infeasible and rescheduling has to be triggered. If this feature of raw material delivery was modelled from the beginning, such infeasibility could have been avoided. With this example other disadvantages of reduced modelling become obvious. Passing the optimized solution of a low-fidelity model to the shop floor, and letting operators decide on how to implement it is less likely to be successful than providing a solution that has been validated using a high-fidelity model.

The case study which is considered in this paper, has also been investigated using a MILP-based decomposition approach in Yfantis et al. (2022, 2019) and a CP-based decomposition approach in Klanke et al. (2021b). However, in these papers, the real process was abstracted substantially and several real-life operating constraints could not be considered. Therefore, the approaches presented in these papers were not accepted by the management of the plant.

To summarize, a high modelling depth is often vital to ensure the feasibility of schedules on the shop floor level. Mathematical or constraint programs, custom schedule builders or simple heuristic scheduling algorithms usually do not generate schedules that satisfy complex constraints as they are present in real-world problems. Therefore, in this contribution, an industrial-strength DES is used for schedule generation in combination with a meta-heuristic, an Evolutionary Algorithm.

SO combines the advantages of high-fidelity simulation and of algorithmic optimization and can take into account the overall execution of the schedule of the system and not only the local situation of the units or of the tasks as it is the case for simple local heuristics. In SO a feedback loop is set up: The optimizer provides inputs (scheduling decisions) to the simulation, the simulation evaluates the schedules and feeds back the objective function values to the optimizer, which uses the feedback to propose new scheduling decisions and the solution quality gradually improves.

The integration of simulation and optimization can be done in many ways, as described in several review articles: See e.g. Smith (2003) and Amaran et al. (2016) for SO for real-valued and combinatorial search-spaces independent of the application area, Negahban and Smith (2014) for a review specifically for manufacturing systems and Figueira and Almada-Lobo (2014) for a review of hybrid approaches and a taxonomy of these.

Meta-heuristics are frequently employed as optimizers for combinatorial optimization problems in the context of SO. Examples are described by Tasoglu and Yildiz (2019) who used Simulated Annealing and the ARENA simulation software, Syberfeldt and Lidberg (2012) who used Cuckoo search and SIMUL8 as the simulator, Yazdani et al. (2021) who use Differential Evolution and an Imperialist Competitive Algorithm as the optimizer or Allal et al. (2021) who use the Ant Colony System as the optimizer and developed the simulator in NetLogo.

EAs are frequently used as a meta-heuristic optimization algorithm for planning and scheduling problems. This also true for non-SO approaches, where an explicit evaluation formula, a heuristic algorithm or a Mixed-Integer Linear Programming (MILP) is used for schedule evaluation (He & Hui, 2007; Reeves, 1995; Wongthatsanekorn & Phruksaphanrat, 2015). Several encoding schemes for allocation and sequencing decisions, as well as mutation and crossover operators, have been proposed (Defersha & Rooyani, 2020; Ishikawa et al., 2015; Urlings et al., 2010). Examples for the use of EA in SO are the works of Azzaro-Pantel et al. (1998), Dong and Medeiros (2012), Istokovic et al. (2020), Lin and Chen (2015), Oyarbide-Zubillaga et al. (2008), Yang et al. (2007), Yazdani et al. (2021).

Dong and Medeiros (2012) optimized a steel pipe manufacturing process using the Tecnomatix Plant Simulation software. An economical objective including changeover, inventory and penalty costs was optimized, using the integrated GA provided by the software. The production sequences, and the release times of the batches were varied. A disadvantage of using the optimization algorithms integrated into the simulation software is that customization for special purposes or more involved problems is not possible.

In Lin and Chen (2015), a semiconductor assembly facility was investigated, the operation of which is similar to a hybrid flow shop. A genetic algorithm that encodes allocation decisions on the line and machine levels is employed as the optimizer, while the simulation software Plant Simulation is used to evaluate the quality of the allocation decisions. Since the problem is modelled non-deterministically, 30 repetitions were performed to evaluate a single allocation.

Only limited work has been published on SO approaches to scheduling problems from the process industries. A seminal paper was published by Piana and Engell (2010), where the production concept of pipeless plants with autonomous guided vehicles was investigated. The challenge here lies in the uncertainty that results from the routing conflicts between the autonomous vehicles, the paths of which cross regularly. Therefore the sequencing decisions where taken by the EA based on average transportation times while the exact schedule was generated by a simulation that included the path planning algorithm. The average transportation times where updated periodically and used to update the scheduling model.

Little work can be found in the literature where allocation and sequencing decisions are considered simultaneously for a scheduling problem of a complexity that requires industrial-strength simulation software. Often, only one of the two degrees of freedom is encoded directly and the other is determined dynamically during simulation or via heuristics, as in Lin and Chen (2015) and Dong and Medeiros (2012).

In Klanke et al. (2021a) the medium-term scheduling of an industrial batch processing plant, that is similar to the processing plant investigated in this work was optimized. Allocation and sequencing decisions were encoded and optimized independently for orders that consisted of up to 7 batches, instead of encoding batches individually. This helped to keep the overall search-space at a tractable size, while implicitly reducing changeover times between orders. In order to obtain high-quality solutions, it was decided to not couple the allocation decisions to the sequences of orders, as this would have made promising allocations inaccessible.

As mentioned above, the problem at hand requires to consider not only allocation and sequencing decisions but also batching or order splitting in order to be able to meet the due dates. In the literature, EAs that decide on batching, sequencing and allocation at the same time, is rarely encountered. In Defersha and Chen (2012) a lot streaming problem of a flexible job shop was investigated. The number and size of sub-lots, allocation and sequencing decisions are integrated into a single encoding. The problem was modelled and simulated as a

mathematical program and because of a low problem complexity, the evaluation times were small, but for case studies of industrial complexity the evaluation times are significantly larger and it is necessary to keep the search space at a manageable size.

Nomenclature

Indices	
o	Base or split order
u	Unit
Sets	
$\mathcal{B} \subset \mathcal{O}$	Subset of customer orders for the batching chromosome
$\mathcal{O}$	Set of customer orders
$\mathcal{O}^s$	Set of base and split orders
$\mathcal{S}_o$	Set of split orders of order o
$\mathcal{U}$	Set of units
Π	Set of permutations
Constants & Parameters	
MR_{ARS}	Mutation rate - Add Random Split
MR_{ARSW}	Mutation rate - Add Random Split Weighted
MR_{CES}	Mutation rate - Change Existing Splits
MR_{PM}	Mutation rate - Point Mutation
MR_{SIM}	Mutation rate - Simple Inversion Mutation
n	Current generation
N	Number of generations
s_{oj}	Relative amount of jth split of order o
$s_{max,o}$	Number of splits of order o
SP_{parent}	Parent selection pressure parameter
$SP_{survivor}$	Survivor selection pressure parameter
$P_{elitist}$	Fraction of the elite solutions
δ	Due date
λ	Number of children
μ	Population size
ρ	Penalty parameter
Miscellaneous	
D	Element-wise difference of samples
$\overline{D}$	Mean value of D
a_i	Individual
P	Parent
p	Probability/p-value
X_i	Sample with configuration i
$\mathbb{U}(0,1)$	Single sample from an uniform distribution between 0 and 1
α	Allocation chromosome/significance level
β	Batching chromosome
π	Sequence chromosome
Objectives	
AAT	Amount Averaged Tardiness
T	Tardiness
Operators	
$a \overset{R}{\leftarrow} \mathcal{A}$	Assign a a random element from set $\mathcal{A}$
$[\cdot]$	Element of a tuple/ordered set/sequence

3. Case study

The case study considered here is an industrial processing plant that consists of two stages, a formulation stage and a filling stage. A transfer panel connects the two stages and with several intermediate buffer tanks. The intermediate buffer tanks are used to increase the overall productivity of the plant by flexibilizing the transfer of intermediates from the formulation lines to the filling lines. In times, when the formulation lines produce faster than the filling lines can handle the incoming intermediates, the intermediate products are stored, while the filling lines are supplied by the stored contents from the intermediate buffer tanks in times, when the formulation lines produce slower than the filling lines. The intermediates either pass through the intermediate storage tanks or are transferred directly from the formulation lines to the filling lines.

In Fig. 1 an abstract representation of the processing plant is shown. The formulation stage consists of seven parallel processing lines, where each formulation line is equipped with a mixing tank (MT) that blends liquids and powders, an intermediate buffer tank (IBT) that feeds a mill and two ripening tanks (RTs), where a final ingredient is dosed and a minimum ripening time has to elapse. Only one ripening tank can be used at the same time, but when several batches have to be produced they are operated alternatingly. From the ripening tanks, the intermediate products may either be transferred to one of the seven buffer tanks or they are directly transferred to the filling stage. When neither a buffer tank nor a filling station is available for storage or further processing, the intermediates can remain inside one of the ripening tanks prior to discharging. Full equipment connectivity is given, i.e. each formulation line may feed each buffer tank and each filling station.

In the seven filling lines, the intermediates are packaged into a variety of final containers. In contrast to the formulation lines, the filling lines work in a semi-continuous manner, i.e. when a filling operation takes place, the source tank (a buffer tank or a ripening tank) is gradually discharged during the filling time, necessitating synchronous operation of the source equipment and the filling line. The operation of the filling lines requires a large amount of human labor. Since there is no night shift, the filling lines are idle at night, which requires that the intermediates are stored in the buffer tanks or the ripening tanks overnight. All charging and discharging operations are performed at constant flow rates. Multiple batches of the same formulated intermediates can be fed to the same buffer tank until the buffer tank reaches its upper capacity limit. In this case, the charging operation from the formulation tank is interrupted until the buffer tank has discharged enough of its content to a filling station and charging can be resumed.

In all of the equipment, sequence-dependent changeover times are defined for cleaning if two subsequent batches, storage operations or filling operations use different raw materials or intermediate products. Within the scope of this work, the processing times of all operations are deterministic, but product-specific and line-dependent in both stages of the process.

In order to provide a basic understanding of the main operations that are required for the production of one batch of a product, a simplified schedule is displayed in Fig. 2. Two orders $O1$ and $O2$ each consisting of a single batch are produced in this schedule. We do not display the other units of a formulation stage, i.e. a *MT*, *IBT* and *Mill*, since this equipment is operated in a simple, sequential fashion. As far as $O1$ is concerned in this example, *RT2* is filled with the intermediate product. After the minimal ripening time has elapsed, the content of *RT2* is discharged into *Buffer Tank 1*. For the purpose of (dis)charging, the operations on both units overlap, i.e. synchronization of operations is necessary. *RT2* is then free to process the next operation, while the discharging of the intermediate to the filling line *Fill 1* can take place from *Buffer Tank 1*. Again, the discharging operation of *Buffer Tank 1* and the filling operation overlap, until all the intermediate is processed. Since the next batch that requires filling comes from $O2$ a cleaning time has to elapse. The same holds for *RT1* where it is assumed that another order was processed before $O2$, which is why the ripening of $O2$ starts after a cleaning operation. The DES enables preemption of tasks and therefore a waiting operation is executed, while Fill 1 is cleaned. After the cleaning finished, $O2$ is filled in *Fill 1* directly from *RT1*. This takes considerably longer than charging *Buffer Tank 1* from a ripening tank

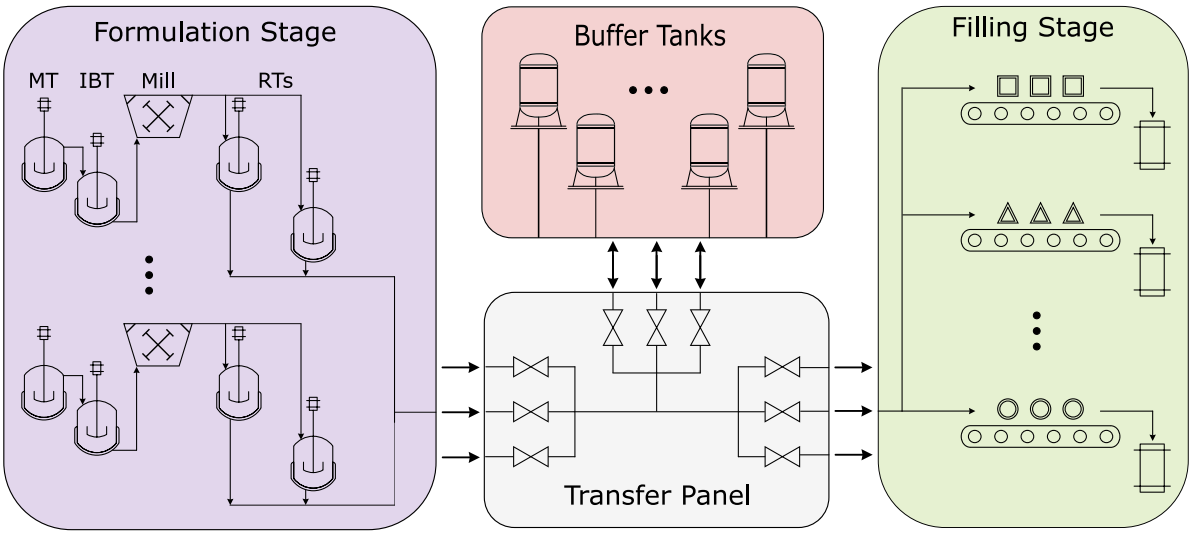

Fig. 1. Schematic representation of the production plant consisting of seven parallel formulation lines, each of which consists of a Mixing Tank (MT), an Intermediate Buffer Tank (IBT), a Mill and two Ripening Tanks (RTs) in the formulation stage, seven buffer tanks, and seven filling lines in the filling stage. A transfer panel connects the two production stages and the buffer tanks.

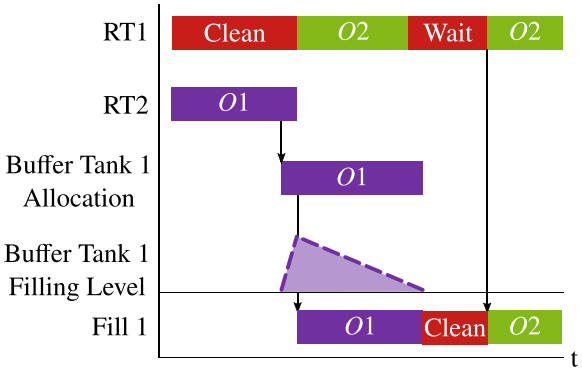

Fig. 2. A simplified exemplary schedule with two orders consisting of one batch each. Arrows indicate the source and destination operation of charging and discharging operations.

Table 1
Due dates and number of batches for each order of the three test cases.

	Order indices o	Due dates δ_o [h]	$\lvert S_o \rvert_{max}$ [a]
Case 1	1,4,7,10,13	264	4
	2,5,8,11,14	168	2
	3,6,9,12,15	72	7
Case 2	1,4,7,10,13	72	5
	2,5,8,11,14	96	3
	3,6,9,12,15	168	5
Case 3	1,4,7,10,13	48	1
	2,5,8,11,14	144	3
	3,6,9,12,15	192	9

[a]Batches per customer order.

as is the case with $O1$. Therefore, $O2$ takes significantly longer on $RT1$ than $O1$ on $RT2$.

Customer orders specify quantities of a demanded final (packaged) product. The customer orders can be divided into several production orders. In most cases, the volume of a production order exceeds the maximum processing capacity of the individual units in the plant and therefore the production orders comprise several batches. For each of these production batches, a specific allocation and sequencing decision can be made. In a typical order structure, for some of the products, it is not possible to meet the due dates without parallel processing of the production orders. For these customer orders, the batching decisions are particularly important. We assume job availability for each batch, i.e. that each batch can proceed to the next processing step after it finished processing in the previous step (Potts & Kovalyov, 2000).

We defined three different test cases to which our approach is applied. In Table 1, the data for the 15 customer orders of each test case, i.e. the due dates and the number of batches per order, is displayed. In total 65 batches are produced in every test case. The type of product, processing times and flow rates are different for every order.

3.1. Discrete-event simulation implementation

A DES treats the plant operations as a sequence of events, where each event marks a change in the state of the system. Between two consecutive events, no state-transitions are assumed to take place but a certain amount of time passes.

Initially, the DES schedules *primary events*, i.e. those for which it is known that they have to be executed. An example is the *start operation* event of an operation for every order. All *primary events* are then executed in chronological order until a stopping criterion is met. The second type of events, denoted *Conditional events*, are only processed, when other events lead to the satisfaction of a given condition. Through the execution of both types of events, complex cascades of events are obtained, i.e. several levels of nested conditional events can be triggered and change the state of the system accordingly. The interested reader finds a more detailed explanation of the main algorithms to process events within the *next-event approach* that is used here in Kemp (2003).

Within the DES used in this work custom events can be introduced as *conditional events*, which we use to fix the sequencing and allocation decisions at runtime.

The initial *operation start* events of all orders are usually generated with the help of static priority rules. Similarly, the allocation of the individual operations that an order consists of to processing units is chosen such that it can start as early as possible. Commercial simulation software often generates schedules based on such relatively simple local priority rules which by nature are myopic and do not lead to optimal decisions (Moser et al., 1992).

4. Methodology

As outlined above, in this work a DES is combined with an EA to ensure feasibility of the schedules when applied on floor-shop level.

Due to the complexity of the production process and to obtain a search-space of tractable size, several scheduling decisions are made by the DES and are not defined in the input received from the optimizer. Local scheduling rules are applied, whenever a decision is not fixed by the input to the simulation. For example, the decision whether or not to use an intermediate buffer tank or to discharge to a filling line immediately from the ripening tanks is not defined by the inputs of the simulation. This routing degree of freedom is determined by the simulator by a heuristic. The EA compensates the drawbacks of the local scheduling rules applied by the DES by a global view on the decisions on the allocation of units, the sequencing and the batching decisions. An EA is preferred over other metaheuristics, as it has been shown to perform well on similar production scheduling problems (Defersha & Rooyani, 2020; Jankauskas et al., 2019; Piana & Engell, 2010).

An overview of the SO approach can be seen in Fig. 3. After a heuristic or random initialization phase, the fitness evaluation of the initial population takes place. For fitness evaluation, the individuals pass their allocation, sequence and batching information to the DES and receive the value of the objective functions for the resulting schedule. Then the termination criterion, in our case a maximum number of generations, is checked, and if the maximum number of generations has not yet been reached, the survivor and parent selection take place. After that, the recombination mechanisms are invoked, which act on all three chromosomes of an individual separately. Thereafter, mutation is applied to the three resulting chromosomes individually. It may be the case that the chromosomes of an offspring individual are not consistent after the application of the recombination and mutation operators. Then repair algorithms are applied to obtain consistent offspring that can be evaluated in the fitness evaluation in the next iteration of the EA. In the following sections, the individual steps of the optimization approach are presented in detail.

4.1. Encodings

We employ three independent chromosomes that make up an individual a_i for each of the three degrees of freedom, i.e. allocation, sequencing and batching. The separation of the allocation and sequencing decisions comes at the price of redundancy, i.e. several genotypes map to the same phenotype, but has the advantages of modularity and extensibility, which is discussed in more detail in Section 4.8.

Due to the combinatorial explosion of the search space, making a sequencing and assignment decision for every element of the problem, i.e. every batch, is not tractable. Hence, aggregating elements to groups and making scheduling decisions for these groups is a measure to decrease the size of the search space. Here, the groups of elements are the production orders that comprise of several production batches. The batches of one order are produced consecutively on the same unit once the first batch of the order has been started on a unit. As mentioned in Section 3, the processing times differ significantly between the units of each stage for both stages of the process. Therefore, the quality of the schedules is very sensitive to the allocation of the production orders to units. As a consequence, a direct encoding of this degree of freedom is introduced and preferred over a heuristic determination of the allocation. The allocation chromosome is defined as a mapping of production orders to units $\alpha \subset \mathcal{O}^s \times \mathcal{U}$.

Following the same argument as for the allocation chromosome, i.e. that decisions cannot be tractably made for every batch and every unit individually, a global, stage-independent sequence $\pi \in \Pi$ is defined, where Π denotes the set of all possible permutations of production orders $\mathcal{O}^s$. Because all production orders are sequenced in π and they may run on different lines, the sequence encoding only influences the relative position of the production orders that run on the same unit. This order is prescribed globally. The details of the encoding of the allocation and sequencing decisions can be found in Klanke et al. (2021a).

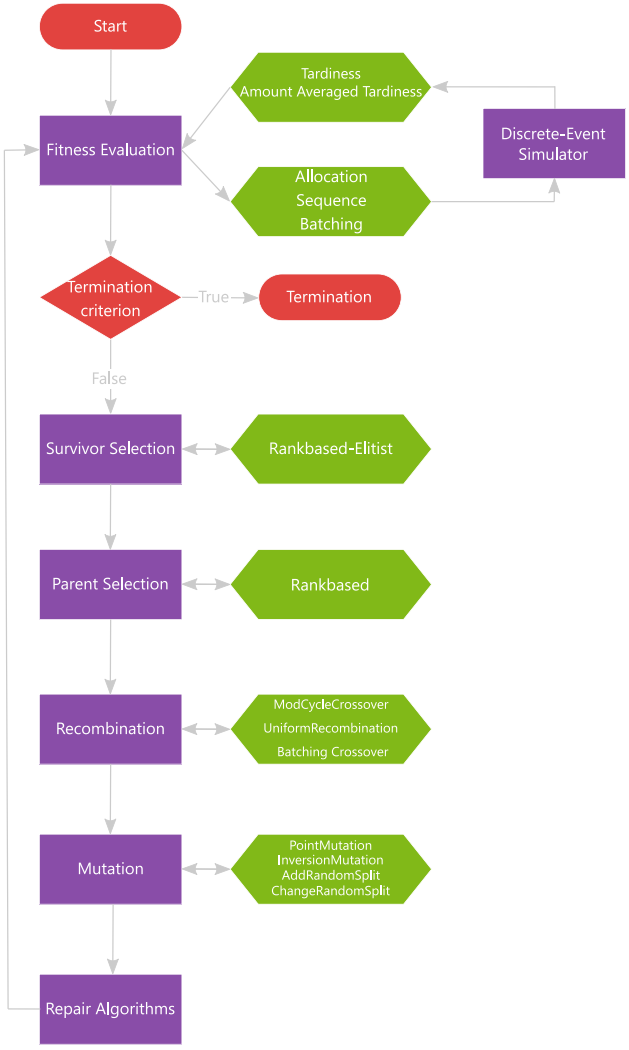

Fig. 3. Schematic representation of the simulation-based Evolutionary Algorithm.

The batching chromosome that is introduced in this work encodes the relative amount and thus indirectly the number of batches for each of the production orders that are derived from a customer order. For each production order, the sequencing and allocation decisions are encoded by the other two chromosomes of the individual. A batching chromosome is defined as $\beta = \{\{o, (s_{o1}, \ldots, s_{oj}, \ldots, s_{o,s_{max,o}})\} \mid o \in \mathcal{B}\}$, where $\mathcal{B} \subset \mathcal{O}$ is the set of all customer orders that are split into two or more production orders of amounts s_{oj}, and $s_{max,o}$ defines the number of production orders into which a customer order o is divided. Production orders that result from a split are denoted as split orders, while production orders that do not result from a split are denoted as base orders. Thus every order o is assigned an s-tuple S_o, that contains the relative amounts of the production orders s_{oj}. To avoid symmetries in the encoding, the amounts s_{oj} are sorted, such that $j < j' \Rightarrow s_{oj} \leq s_{oj'}$. The set of production orders is obtained from a batching chromosome and the set of customer orders, i.e. $\mathcal{O}^s = f(\beta, \mathcal{O})$. $\beta = \emptyset$ implies that $\mathcal{O}^s = \mathcal{O}$, i.e. that all customer orders are treated as production orders. An example of a batching chromosome is, $\{\{O1, (0.5, 0.5)\}, \{O2, (0.2, 0.3, 0.5)\}\}$, where the base order $O1$ is split into two split orders $S_{O1} = \{O1a, O1b\}$ each producing 50% of the total demand of order $O1$ and $O2$ is split into three split orders $O2a$, $O2b$ and $O2c$ producing 20%, 30% and 50% of the total demand of order $O2$. In the simulation, the relative amounts are mapped to an integer number of batches by a simple heuristic.

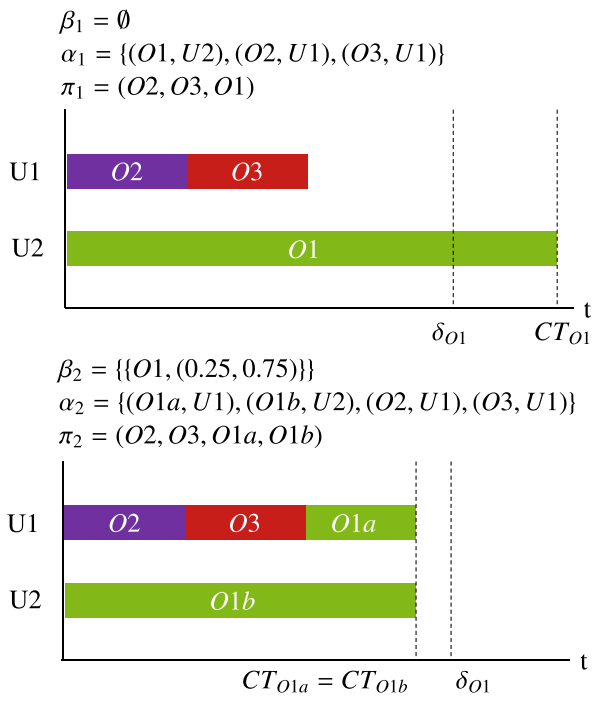

Fig. 4. Two exemplary encodings and the resulting schedules. In the first schedule, the encoding does not contain a split into production orders and the due date of order $O1$ is violated. In the second schedule, order $O1$ was split, such that the due date can be satisfied.

The introduction of the batching chromosome has far-reaching consequences for the consistency of the genomes of the individuals and for the genetic operators. The set of production orders $\mathcal{O}^s$ depends on the batching chromosome. Therefore, when new splits are added, the allocation and sequence chromosomes of different individuals encode decisions for different sets of production orders and variable-length chromosomes result.

Each individual is therefore defined by its set of chromosomes, $a_i = [\alpha_i, \pi_i, \beta_i]^T$. Symmetries in the presented encoding occur when two orders that are adjacent in the sequence chromosome are produced on different units. In that case, swapping the two orders in the sequence leads to the same schedule.

An example of the three encodings that shows how the batching decision facilitates meeting due dates can be seen in Fig. 4. In the upper schedule, $O2$ and $O3$ are assigned to $U1$, as encoded by α_1, where $O2$ is processed before $O3$ due to their order of appearance in π_1. As $O1$ has a large total processing time, a violation of its due date δ_{O1} occurs. In the lower schedule, a split of $O1$ is introduced, leading to two orders $O1a$ and $O1b$, that substitute $O1$ in α_2 and π_2. In this example, the 25% split of $O1a$ is assigned at the end of $U1$, finishing in time, while the remaining 75% remain on $U2$, also finishing in time.

4.2. Crossover operators

Due to the introduction of the batching chromosome, in the recombination step chromosomes of variable-lengths have to be recombined. This requires extensions of the crossover operators for the allocation and sequence chromosomes, which are presented in this subsection first. Then a new recombination operator for the batching chromosome is proposed.

We include all those production orders of the parent chromosomes that occur in one of the parents in the child chromosome. If the first parent encodes decisions for $\mathcal{O}^s_1$, i.e. a set whose elements are base and split orders, and the second parent encodes decisions for $\mathcal{O}^s_2$, then the child chromosome after recombination encodes decisions for $\mathcal{O}^s_3 = \mathcal{O}^s_2 \cup \mathcal{O}^s_1$, i.e. it is possible that the child contains split and base orders of the same customer order at the same time. We chose this approach to be able to design the crossover operators without the information on the concrete set of orders which are split according to the batching chromosome. In the repair algorithms (Section 4.4), however, this information is available and redundant base or split orders are removed and a consistent chromosome is restored.

In the case of the recombination operator for the allocation chromosome, we propose a modified uniform crossover operator (Algorithm 1, see Appendix). For each order o, the child allocation α_3 is assigned the allocation either from $P1$ or $P2$ with equal probability, if the order o exists in both parents. For each order, that is only present in either of the parents, the allocation from that parent is added to α_3.

In the case of the crossover operators for the sequence chromosomes, the design of an operator is more involved. The crossover operators that can be found in the literature are applicable to chromosomes of equal length that contain identical sets of elements. In our case, due to the different batching decisions that have been taken for different individuals, the lengths of the chromosomes and the elements contained in them may differ in a pair of parent chromosomes. We chose to extend the Cycle Crossover, as it has shown to slightly outperform the OX1 crossover operator on our case study instances in preliminary studies. Algorithm 2 (see Appendix) defines a Modified Cycle Crossover (MCC). Like the Cycle Crossover, it consists of two rounds, the first of which yields a cycle in the two parent chromosomes and the values which are inherited by the offspring (Algorithm 2, lines 9–31). However, during the first round it may occur that the partial cycle points towards an element in parent $P2$ that does not occur in the other parent $P1$. In this case the index pointing to the element in $P2$ is advanced by one position. If this new index points to an empty position in π_3, the element $\pi_2[pos_{P2}]$ is inserted into the offspring chromosome. This way, the relative position of $\pi_2[pos_{P2}]$ in the offspring chromosome is maintained approximately. Then a new random index pos_{P2} is chosen, from which the cycle is continued. Before the second round, all yet missing elements, i.e. elements, that occur in $P1$ or $P2$ but not in the offspring, are collected alternatingly in their order of appearance in π_{miss} (denoted by $zip()$ in Algorithm 2, line 33). The insertion starts with $P1$. In the second round, all missing elements are inserted into π_3 in their order of appearance in π_{miss}.

An example of the application of Algorithm 2 can be seen in Fig. 5. The MCC starts from parent permutations $\pi_{P1} = (O2, O1, O4, O3)$ and $\pi_{P2} = (O4a, O4b, O3, O1, O2)$ in the 1st round. At first a random element of π_{P1} is chosen, in this case $\pi_{P1}[1] = O2$. It is inserted into the child chromosome and the value in $P1$, which is identical to that in π_{P2} at the same index is searched. In this case, element $O4a$ is searched for in π_{P1}, but it does not exist. Therefore, the index of $P1$ is incremented by one, the element $O4a$ is added to π_3 at index 2, and the procedure is restarted from a random index, in this case $pos_{P2} = 3$. Since $O3$ has index 4 in $P1$ and no element is inserted at that index in π_3, $O3$ is inserted in the child chromosome at that position. The index pos_{P2} is set to the same value as pos_{P1}, i.e. to 4, and the element in $P1$ that is equal to $\pi_{P2}[4]$ is searched. This element is $\pi_{P1}[2] = O1$. Since $\pi_3[2]$ is already filled, the first round ends.

In the second round, all those elements in $P1$ and $P2$ that are not yet elements of π_3 are alternatingly inserted into the remaining positions of π_3. Those elements are inserted in ascending order of their source indices in either of the two parents, i.e. $O4b$ is inserted before $O4$. Elements from $P1$ are prioritized in the case of identical indices, i.e. $O1$ from $P1$ is inserted before $O4b$ from $P2$. The result of the second round contains the customer order $O4$, as well as the production orders $O4a$ and $O4b$, from which either alternative 1 or alternative 2 are used as offspring sequence chromosome after the application of the sequence repair algorithm (Algorithm 3, see Appendix).

For the crossover operator of the batching chromosome, it has to be noted that the batching chromosome usually only contains a subset of all customer orders $\mathcal{B} \subset \mathcal{O}$, which means that the set of possible splits

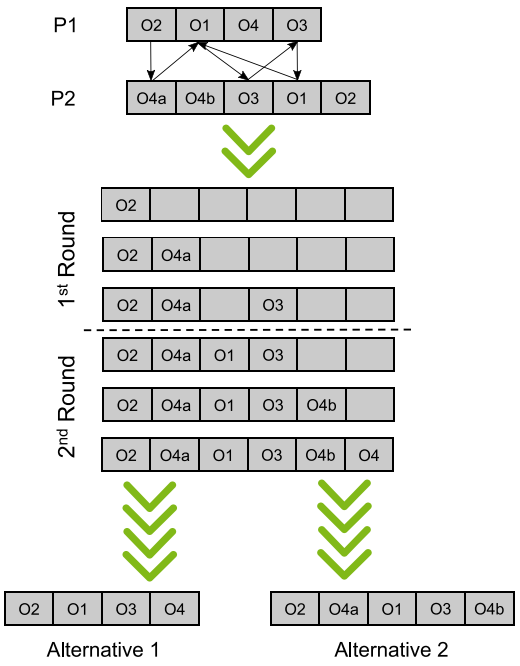

Fig. 5. Schematic representation of the steps of the Modified Cycle Crossover for the sequence chromosome.

encoded by the offspring individual generated in recombination is $\mathcal{B}_3 = \mathcal{B}_2 \cup \mathcal{B}_1$. In order to prevent an uncontrolled growth of the search space during the optimization, orders $o_1 \in \mathcal{B}_1 \setminus \mathcal{B}_2$ and $o_2 \in \mathcal{B}_2 \setminus \mathcal{B}_1$ only have a 50% chance of being inherited by the offspring, but splits that occur in both parents are guaranteed to be passed on to the offspring individual. Algorithm 5 (see Appendix) gives the details of the crossover operator of the batching chromosome.

4.3. Mutation operators

The mutation operator for the allocation chromosome is a Point Mutation operator that randomly chooses a production order o from the allocation chromosome and assigns a new, eligible unit to that order. For the sequence chromosome, the Simple Inversion Mutation operator, as described in Larranaga et al. (1996), is used. For both operators, no changes are required by the introduction of the batching chromosome. For the batching chromosome, no standard mutation operators are available in the literature and so they had to be newly designed. Reachability, i.e. the ability of mutation operators to transfer the state of the chromosome into any other state in a finite number of steps (Beyer & Schwefel, 2002), is one of the key requirements in the design of mutation operators. Therefore three different operators *AddRandomSplit* (ARS), *AddRandomSplitWeighted* (ARSW) and *ChangeExistingSplit* (CES) were designed for the introduction of new splits and for changing the amounts per split. This enables to have an arbitrary number of split orders, down to the level of individual batches, and an arbitrary distribution of the number of batches to the different split orders.

The *AddRandomSplit* and *AddRandomSplitWeighted* operators, the pseudocode implementation of which can be seen in Algorithm 6 (see Appendix), only differ with respect to the probability that a certain order is selected for splitting. In the former operator, all orders have the same probability of being selected. With the latter operator, the historical average tardiness of all evaluated individuals is used as weight for the selection probability, i.e. a higher historical average tardiness increases the chance of an order to be split. This follows the idea that orders where a larger tardiness was observed throughout the optimization are more likely to benefit from a parallel execution on different units.

The *ChangeExistingSplits* mutation operator is applied after one of the other two batching mutation operators has been applied. Its pseudocode implementation can be seen in Algorithm 7 (see Appendix). This mutation operator picks a random split order o and selects two of its split amounts a_{inc} and a_{dec}, to be increased and decreased by a fixed amount. For example, the split order $\{O1, (0.5, 0.5)\}$ becomes $\{O1, (0.4, 0.6)\}$ after application of the *ChangeExistingSplits* mutation operator.

4.4. Repair algorithms

To deal with inconsistent genomes, a repair algorithm for the allocation and for the sequence chromosome is needed. The batching chromosome and its mutation and crossover operators are designed such that batching chromosomes are always consistent. Inconsistent chromosomes are encountered in two situations:

- Missing splits: When new splits are introduced by a mutation of the batching chromosome, the corresponding production orders are not encoded by the sequence and allocation chromosome.
- Redundant splits: The crossover operators for the allocation and sequence chromosomes introduce both a base order, e.g. $O1$ from parent $P1$, and split orders, e.g. $O1a$ and $O1b$, from parent $P2$ into the child chromosomes.

In Algorithm 4 (see Appendix), in lines 1 and 2 the missing production orders and redundant base and production orders are gathered. All missing production orders are then inserted into the chromosome and are assigned a random (eligible) unit. The allocations for redundant split or base orders are simply removed from the chromosome. Similarly, to repair sequence chromosomes, in Algorithm 3 (see Appendix) missing production orders are added at a neighbouring position of production orders that belong to the same customer order and redundant base or production orders are simply removed from the chromosome. When a batching chromosome encodes a split of an order, but an allocation or sequence chromosome already contains more split orders than are actually necessary, the excess splits are removed. Those that are required according to the batching chromosome remain in the repaired allocation or sequence chromosome.

An example of how the two repair algorithms work can be seen in Fig. 6. A batching chromosome, splitting order $O4$ at 50 %, is considered. In the case of the allocation chromosome, the allocation decision for the first production order $O4a$ is set to be the same as for the customer order $O4$ and the allocation decision for the second production order $O4b$ is set to the randomly chosen, eligible unit $U3$. For the sequence chromosome, the result of the second round of the MCC from Fig. 5 is considered. Since the batching chromosome encodes that $O4$ is split, the customer order $O4$ is simply removed from the sequence chromosome. In this way, a consistent set of chromosomes is obtained from initially inconsistent chromosomes.

4.5. Objectives

In this work, we consider two different objectives. The ultimate goal is to minimize the cumulated tardiness of the orders under consideration:.

$$T = \sum_{o \in O} max(0, CT_o - \delta_o) = \sum_{o \in O} max(0, \max_{o' \in S_o}(CT_{o'}) - \delta_o) \quad (1)$$

The definition of the tardiness T was adapted to account for the batching decisions, by setting the tardiness of order o to be equal to the maximal tardiness of its split orders, in case it has been split.

However, we observed that the minimization of the cumulated tardiness objective does not provide enough guidance to make reasonable batching decisions. This happens because many newly introduced splits do not reduce the tardiness value as long as other decisions do not change the allocation or sequence. However, after a split for a given

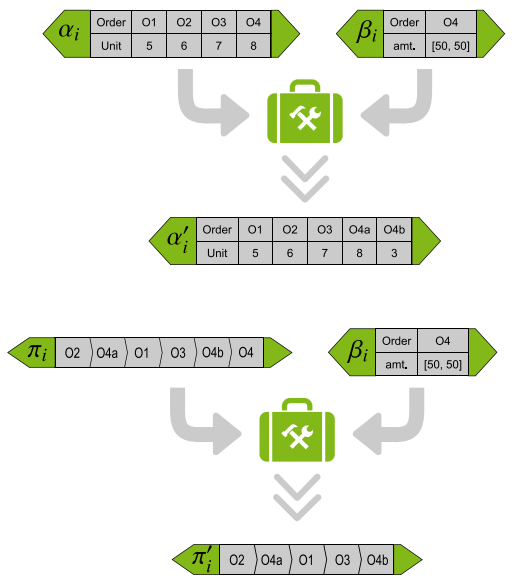

Fig. 6. Schematic representation of the repair of an allocation and a sequence chromosome.

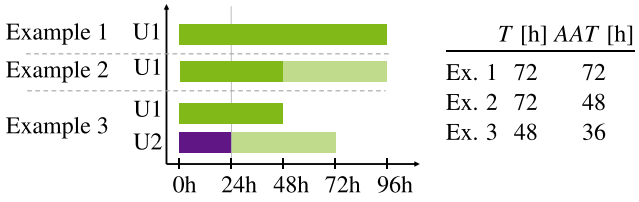

Fig. 7. Schematic representation of the differences of the Tardiness and the AAT objective. In Example 3 another order is forcing the split order to start at 24 h.

order, there is a bigger potential for further tardiness reduction than if the order is not split. For this reason, we define the additional objective Amount Averaged Tardiness (AAT), with $AAT \leq T$ holds:

$$AAT = \sum_{o \in O} \sum_{o' \in S_o} w_{o'} \cdot max(0, CT_{o'} - \delta_{o'}) \tag{2}$$

In the definition of AAT we use weights $w_{o'}$ that represent the relative total amount of production order o', $\sum_{o' \in S_o} w_{o'} = 1$. The objectives T and AAT are equal when no splits have been introduced.

In Fig. 7 an example is shown how the AAT relates to the T objective. In *Example 1*, a single order o is delayed and finishes 72 h after its due date. Since no split orders exist, $w_{o'} = 1$ and $T = AAT = 72$ h. In *Example 2* order o is split into two orders, the first of which is 24 h and the second of which is 72 h delayed. Since the order is split in half, $w_1 = w_2 = 0.5$. The tardiness value remains unchanged at 72 h, while the AAT value is reduced 0.5×24 h $+ 0.5 \times 72$ h $= 48$ h. In the third case, an order is split and allocated on two different units $U1$ and $U2$, where $U2$ is already in use for the first 24 h. The tardiness value amounts to 48 h, taking into account only the larger of the two tardiness values, while the AAT amounts to 36 h.

By optimizing the AAT instead of the tardiness, we incentivize the optimizer to keep splits that may not improve the tardiness value immediately but may lead to an improvement later on. For example, if one of the two splits from *Example 2* is allocated to another unit, a schedule as in *Example 3* can be obtained.

Total tardiness is used as the true measure of the solution quality since the final products are only available for delivery when the total demand of a customer order has been fulfilled. The AAT is only used as an auxiliary objective.

4.6. Parent and survivor selection

For the survivor selection, we use a $(\mu + \lambda)$ selection (Beyer & Schwefel, 2002). The selection operator is the rank-based roulette wheel selection, as described in Eiben and Smith (2004). Compared to the standard roulette wheel selection, it has the advantage that a set of solutions with similar objective function values can still be assigned different probabilities of selection. This is useful in situations, where the spread between the high- and low-quality solutions in a population is small. This selection scheme uses a selective pressure parameter $1 \leq SP_{survivor} \leq 2$. A high value of SP gives more weight to high-quality solutions, whereas with $SP_{survivor} = 1$, all solutions are selected with equal probability. It was observed that in many cases, promising solutions do not survive for more than a few generations, even when setting $SP_{survivor} = 2$. To better exploit the traits of these solutions in recombination, it is desirable that these solutions survive as long as no better solutions are found. Hence, we extend the survivor selection procedure by an elitist selection, the fittest $P_{elitist} \cdot \mu$ individuals are selected with certainty and from the remaining $(1 - P_{elitist}) * \mu$ individuals, the survivors are selected according to the rank-based roulette wheel selection.

The parent selection only uses the rank-based roulette wheel selection. The corresponding tuning parameter is denoted SP_{parent}.

4.7. Initialization strategies

In the literature, it has been proposed to take the batching decisions prior to solving the allocation and sequencing problem, i.e. a complete decoupling of the two problem settings takes place (Schwindt & Trautmann, 2000). Each such batching decision could be used to initialize the batching chromosomes in the EA proposed in this work. We here use a simple batching strategy for initialization. The 5 orders with the largest average production time on the eligible units are split into two split orders of equal relative amounts. In the results section, we analyse, whether this simple batching initialization heuristic, denoted *Prebatching*, is beneficial to the overall optimization.

For the allocation chromosome, we use an initialization procedure that balances the total allocated processing time across the formulation lines. The procedure is inspired by the *random permutation* method proposed in Chiang and Lin (2013).

For the sequence chromosome, arbitrary static dispatching rules (i.e. rules based on properties that are known prior to simulation) can be used for initialization. In our case, the Earliest Due Date (EDD) dispatching rule is used.

4.8. Discussion of the encoding scheme

A distinctive feature of our approach is the fact that the encodings of the scheduling problem are treated independently in initialization, mutation and recombination. While such an independent treatment has been proposed for allocation and sequencing encodings in the literature (e.g. Ishikawa et al., 2015), when the batching degree of freedom is encoded to our knowledge only integrated encodings have been proposed (e.g. Defersha & Chen, 2012).

Fig. 8 displays the path of the allocation, sequence and batching chromosomes through the recombination step, the mutation step and the repair algorithms. Each pair of parent individuals pass their allocation, sequencing and batching chromosomes, to the associated recombination operator. The resulting children individuals pass their chromosomes to three chromosome-specific operators (in the case of the batching chromosome several operators) of which either one or a subset are applied. The mutated chromosomes are then passed to the two repair algorithms that ensure consistency between the three chromosomes, for which information from the batching chromosome is required. The three resulting consistent children chromosomes are then passed to the fitness evaluation as shown in Fig. 3.

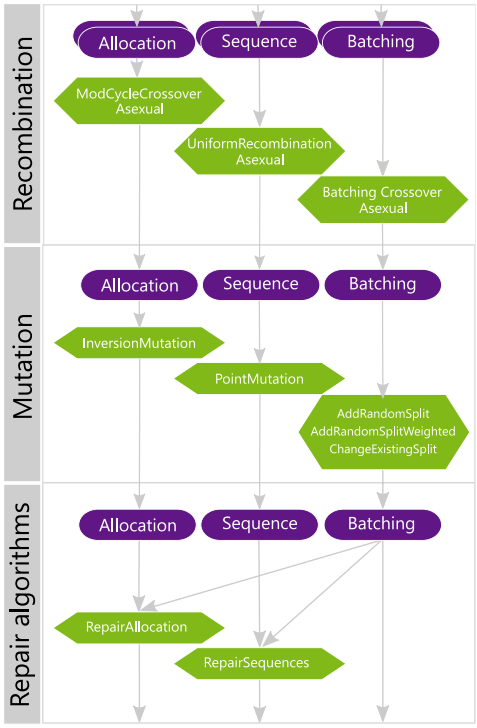

Fig. 8. Detailed representation of the recombination and mutation operators and the repair algorithm. Each arrow carries the information of one or two of the allocation, sequence and batching chromosomes.

An integrated representation was not targeted, as our goal was to obtain a modular and flexible approach and an implementation with not too tightly coupled elements, where encodings, can be added, omitted or substituted by heuristics or other encodings if desired. On the one hand this design satisfies the requirements for industrial use and industrial offerings, where maintainability and extensibility are among the key requirements. On the other hand the optimization can easily be adapted to further production scheduling cases. While there is "No Free Lunch" for any single meta-heuristic algorithm (Wolpert & Macready, 1997), decreasing the workload of a software solution provider through a flexible design allows them to spend more time on the customization to a new problem until a satisfactory performance is obtained.

Taking the batching encoding as example, in the course of the development of the proposed approach, no changes had to be made to the initialization and mutation operators of the allocation and sequence encodings when the batching chromosome was added.

However, of course the interactions between the different chromosomes must be taken into account. In the proposed optimization approach, the repair algorithms ensure that consistent sets of chromosomes result.

5. Results

This section is structured as follows: At first, the real- and integer-valued hyper-parameters that have been chosen in the configuration of the EA are presented. Then, an exemplary schedule for Case 1 is shown that provides insights into the intricacies of the scheduling problem. This example highlights the importance of the batching decisions. After that, the results of a factorial experiment that is used to analyse the influence of the choice of the objective function, of the batching mutation operators, of the heuristic initialization for the batching chromosomes and of the crossover operators are presented. These experiments are done for Case 1. We then provide results on the reproducibility of the optimization approach. Finally, a performance comparison of the proposed optimization approach with and without the batching chromosome and with random search is discussed. The reproducibility analysis and the performance comparison are conducted for all three cases from Table 1.

All computations were conducted on a FUJITSU PRIMERGY RX300 S8 with two 12-core Intel Xeon E5-2697 processors (2.70 GHz) and 192 GB of RAM. The optimizer was implemented in C#7.3 on the .NET framework Version 4.6.1. As the discrete event simulator, INOSIM 13.0.4 Runtime Edition provided by Inosim Software GmbH, Dortmund, Germany was used.

Table 2
Fixed parameter values used in the tuning of the categorical optimization parameters.

Parameter	Value
μ	20
λ	20
SP_{parent}	1.5
$SP_{survivor}$	1.8
$P_{elitist}$	0.4
MR_{PM}	0.3
MR_{SIM}	0.3
MR_{ARSW}	0.7
MR_{ARS}	0.7
MR_{CES}	0.7
ρ	10^8

5.1. Configuration of the algorithm — real- and integer-valued parameters

In Table 2 the parameters chosen for the EA are presented. We refrained from an exhaustive tuning of all real- and integer-valued parameters, as for most of them literature values are available or, as in the case of μ and λ, reasonable values can be deducted directly from the tests. The parameters μ and λ are chosen such that the computational resources are fully utilized. We chose $\mu = \lambda = 20$, because this has shown to be the smallest value where the total capacity of the processors is utilized. To satisfy computational time constraints that were indicated by the end users of the solution, quick convergence is desirable. Therefore we set $SP_{survivor}$ to 1.85 and $P_{elitist}$ to 0.4 as in Klanke et al. (2021a). We set SP_{parent} to 1.5, slightly higher than reported in Razali and Geraghty (2011) for the survivor selection to favour the selection of well-performing individuals. We set the mutation rates of the mutation operators for the sequence and allocation chromosomes MR_{PM} and MR_{SIM} to 0.3. We ran preliminary experiments where these two mutation rates were varied, but we could not observe a significant impact. This is different for the mutation rates of the batching chromosome mutation operators *AddRandomSplitWeighted* (MR_{ARSW}), *AddRandomSplit* (MR_{ARS}) and *ChangeExistingSplits* (MR_{CES}), where the three different levels of $0.3, 0.7$ and 1.0 were tested and 0.7 yielded the best solutions quality on average. Since at most one splitting decision is made or changed per application of the operators, the small exploratory capacity of the operators is offset by a high mutation rate which for the batching chromosome mutation operators is the probability of the operator being applied.

The penalty parameter ρ is added to the objective function value in the case of aborted simulations. The fitness value is then guaranteed to be larger than the fitness value of any feasible individual. Simulations abort when too many orders are allocated to the same formulation or filling line and therefore do not finish within the maximum simulation time horizon.

5.2. Configuration of the algorithm — categorical parameters

As pointed out in the previous section the real- and integer-valued optimization parameters were chosen within ranges proposed in the

Table 3
Results for the tardiness value in hours of three runs with different settings. The best configuration is written in bold.

#	S/AS[a]	$ARS/\ ARSW$[b]	$P/\ NoP$[c]	T/AAT[d]	Run 1	Run 2	Run 3	Avg.	Std. dev.
1	S	ARSW	P	AAT	280.28	288.31	288.55	285.71	4.71
2	S	ARSW	NoP	AAT	255.15	239.88	250.63	248.55	7.85
3	S	ARS	P	AAT	405.38	354.41	305.37	355.05	50.01
4	S	ARS	NoP	AAT	274.51	260.86	228.51	254.63	23.62
5	S	ARSW	P	T	320.40	313.35	271.61	301.79	26.37
6	S	ARSW	NoP	T	304.01	283.51	250.29	279.27	27.11
7	S	ARS	P	T	271.75	358.78	330.93	320.49	44.45
8	S	ARS	NoP	T	246.74	220.52	274.51	247.26	27.00
9	AS	ARSW	P	AAT	291.47	307.78	285.00	294.75	11.74
10	AS	ARSW	NoP	AAT	210.63	210.63	278.94	233.40	39.44
11	AS	ARS	P	AAT	358.38	406.44	390.64	385.15	24.50
12	AS	ARS	NoP	AAT	253.13	303.71	249.28	268.71	30.38
13	AS	ARSW	P	T	275.97	276.80	285.71	279.49	5.40
14	**AS**	**ARSW**	**NoP**	**T**	**209.50**	**248.75**	**208.37**	**222.21**	**23.00**
15	AS	ARS	P	T	285.90	262.94	319.36	289.40	28.37
16	AS	ARS	NoP	T	281.70	212.75	274.51	256.32	37.90

[a] S = Sexual, AS = Asexual
[b] ARS = RandomSplit, $ARSW$ = RandomSplitWeighted
[c] P = Prebatching, NoP = NoPrebatching
[d] AAT = AmountAveragedTardiness, T = Tardiness

literature. In contrast, for several categorical parameters no guidance from the literature is available, which is why the best configuration for those parameters was identified via factorial experiments that were performed for Case 1. The results are presented in Table 3. In total 4 influential categorical parameters have been investigated. For each of the following settings, three repetitions have been made:

1. The use of *sexual* versus *asexual* recombination.
2. The use of the *AddRandomSplit* versus the *AddRandomSplitWeighted* mutation operator.
3. The use of the *Prebatching* initialization strategy.
4. The optimization of the Tardiness criterion directly versus the optimization of the AAT.

All optimization runs of the factorial experiments were conducted with the parameters given in Table 2 and with the total number of generations $N = 50$ and involved the optimization of the allocation, sequencing and batching decisions at the same time. A single evaluation of a schedule takes approximately 9 s, therefore a single optimization run with $N = 50$ takes around 2.5 h.

The sexual recombination operators used in the setting *sexual* are the three operators introduced in Algorithms 1, 2 and 5, while for the setting *asexual*, the recombination operator is that the child chromosomes are a copy of the chromosomes of one of the two parents chosen at random. In case of asexual recombination, consistency of the resulting child chromosome is guaranteed. This way the use of the repair algorithm, becoming necessary through sexual recombination and the mutation, and the less frequent use of the repair algorithm only after the mutation in case of the *asexual* setting, can be compared.

The attainable solution quality of the proposed optimization approach depends strongly on the chosen configuration, as the best (#14) and the worst configuration (#11) deviate by up to 73 % with respect to their average values.

In order to establish statistical significance on the influence of each of the configuration parameters individually, a Wilcoxon test on the data from Table 4 was conducted. For each pair of categorical parameters, the complete dataset was split into two subsets X_i and X_j, e.g. all data generated with sexual recombination are assigned to X_S and all data generated with asexual recombination is assigned to X_{AS}. In Table 4 $\overline{D}_{ij}$ denotes the mean value of the element-wise difference $D_{ij} = X_i - X_j$, W_{ij} the test statistic value and p_{ij} the p-value of the statistical test. The Wilcoxon test was conducted as a two-sided test with a significance level of $\alpha = 0.05$. The Nullhypothesis of this test is that the median value of D_{ij} is not significantly different from zero. Considering the calculated p-values, we can reject the Nullhypothesis

Table 4
Mean difference $\overline{D}_{ij}$, test statistic W_{ij} and p-value p_{ij} for four paired sets of samples of the Wilcoxon test.

$X_i - X_j$	$\overline{D}_{ij}$	W_{ij}	p_{ij}
$S - AS$[a]	7.91	119	0.39
$ARS - ARSW$[b]	28.98	69	0.02
$AAT - T$[c]	16.21	119	0.39
$P - NoP$[d]	62.68	0	1.19E−07

[a] S = Sexual, AS = Asexual
[b] ARS = RandomSplit, $ARSW$ = RandomSplitWeighted
[c] AAT = AmountAveragedTardiness, T = Tardiness
[d] P = Prebatching, NoP = NoPrebatching

for the pairs $ARS/ARSW$ and P/NoP. So the *AddRandomSplitWeighted* operator is a significantly better mutation operator. This is not surprising, as more information than for the *AddRandomSplit* operator is taken into account, however the strong influence is remarkable. For the pair P/NoP the test is even more unambiguous. Not in a single run, in which the orders were pre-batched, a lower tardiness value was obtained than in a run with the same hyper-parameters, but without pre-batching. When pre-batching is applied, 20 production orders instead of 15 have to be sequenced and allocated from the beginning which inflates the search-space. Another explanation for this effect could be that the repair algorithm assigns production orders to different, but randomly chosen units, leading to a low solution quality in the beginning.

For the pairs AAT/T and S/AS the Nullhypothesis cannot be rejected, although $D_{AAT,T}$ and $D_{S,AS}$ indicate that using *asexual* recombination and using the tardiness directly as the objective function is a favourable setting.

5.3. Exemplary schedule

Fig. 9 displays an exemplary schedule for Case 1. In the schedule, the 7 formulation lines *Form 1* to *Form 7*, are represented by the two parallel ripening tanks *RT1* and *RT2*, 7 storage tanks *Buffer Tank 1* to *Buffer Tank 7* and the seven filling lines *Fill 1* to *Fill 7* are visible. Although present in the simulation model, we omitted the Mixing Tank, Intermediate Buffer Tank and the Mill in the schedule for the sake of a clearer presentation. Each block represents one or multiple operations, such as filling, reaction or discharging steps of a production order and each colour represents a distinct production order. In contrast to Fig. 2, cleaning and waiting operations are integrated in the blocks of the plant operations. Due to our encoding, which represents production orders

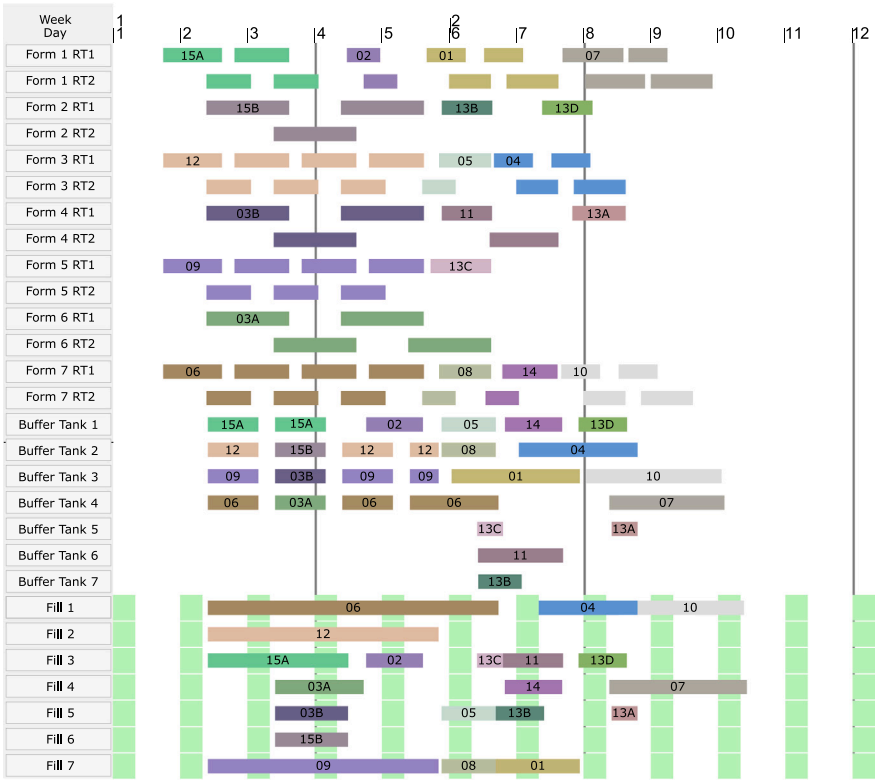

Fig. 9. Exemplary schedule of the case study including all formulation units, the buffer tanks and the filling lines, as well as the due dates of the orders.

rather than individual batches, on the formulation lines only blocks where the batches of a production order are contiguously produced are obtained. This means, that as soon as a certain order is started on a line, no batch from another order can be started before all batches from the already started order are finished.

The green areas represent the nightly downtimes of the filling lines. In addition, the due dates of the orders, either 04.01, 08.01 or 12.01, are displayed with thick, grey vertical lines.

As far as the allocation decisions are concerned, the unit-dependent processing rates of the units and the limited flexibility of the orders render allocations favourable, where units with a high processing rate are utilized more.

In this example, there are 5 orders with a tight due date (Orders 1, 4, 7, 10, 13). Indeed these due dates are too tight to meet all of them. However, a decrease in tardiness is possible, when some of these orders are split by the optimizer and allocated to all 7 filling lines in the beginning. In this example, orders $O3$ and $O15$ have been split into the production orders $O3a$, $O3b$, $O15a$, $O15b$ with 4 and 3 batches each. This leads to a more balanced utilization of the filling lines by the urgent orders. The orders with the loosest due dates are produced last on the filling lines with the exception of $O13C$. However, since the more urgent order $O11$ does not exhibit tardiness, no adversarial effects result from this scheduling decision.

Therefore it can be concluded that the EA provided a reasonable schedule.

5.4. Reproducibility

Avoiding premature convergence of the EA is one of the design requirements in order to obtain reproducible results in spite of the stochastic nature of the approach. Since for the results in Table 3 only three repetitions per configuration were performed only for Case 1, it is unclear whether the standard deviation of the best-in-run tardiness values is acceptably low for cases that the configuration was not tuned for. Also for parameter tuning, only 50 generations were evaluated within $\approx$ 2.5 h. We conducted 5 repetitions, each evaluating $N = 200$ generations to check whether the EA yields reproducible results with a small standard deviation for all three test cases. This, with the given version of the simulator software, corresponds to computing a new schedule overnight which is a common approach in industry.

The best configuration from Section 5.2, #14, was chosen for this analysis. The resulting evolution of the tardiness over the generations can be seen in Fig. 10.

From Fig. 10 it can be seen that all runs within a test case converged to solutions of similar quality although the initial solution quality is significantly worse in all cases. In Fig. 10(a), the average of the best tardiness values from every run after 200 generations is 215.94 h, with a standard deviation of 8.43 h. This leads to a coefficient of variation of 3.90%, which is satisfactory given the complexity of the scheduling task. Similarly, for Case 2 and 3 the standard deviations are 16.67 h and 12.75 h. Note, that the average of the best tardiness values of Case 3 is 33.06 h, hence much smaller than for Case 1 and 2, which leads to a large relative variation while the difference between the best and the worst run, after 200 generations for Case 3 is in between that of Case 1 and 2. For Case 2, the best tardiness values in the initial population (Generation 0) are larger than those of Case 1 and 3. Case 2 has the tightest due dates. This provides an explanation for the larger standard deviation of Case 2.

5.5. Comparison of different approaches

In this subsection, the efficiency of the proposed optimization approach with the batching chromosome over simple random search, the default solution provided by the DES, and over an optimization without the batching chromosome is demonstrated. In Table 5, the results of 5 independent runs of random search, of optimizations with batching chromosome and of optimizations without the batching chromosome are displayed as well as the default results produced by the DES that uses internal heuristic scheduling rules without optimization. For the optimizations, configuration #14 from Table 3 either with or without

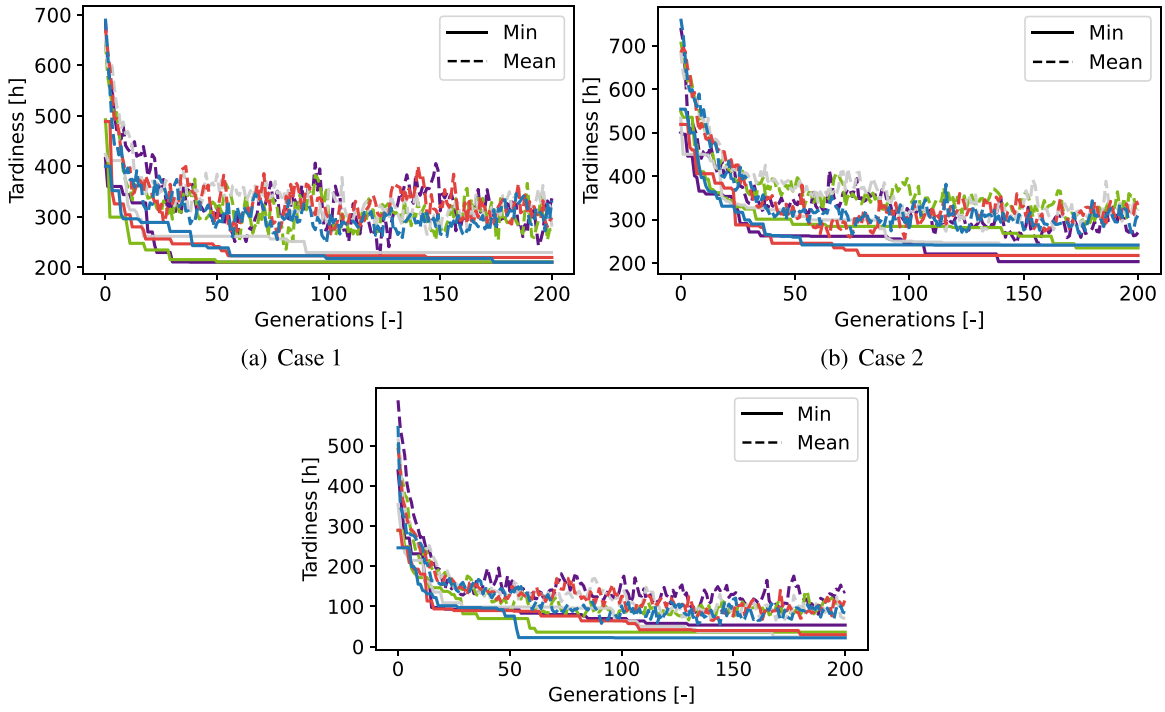

Fig. 10. Reproducibility results for configuration #14 on the three test cases. The minimal and average tardiness value of the population in each generation is displayed. Each distinct colour corresponds to an independent run of the EA.

the batching chromosome, and therefore with or without the $ARSW$ mutation operator, was chosen.

In all cases except for the default schedule 1020 ($= N \cdot \lambda + \mu$) evaluations of schedules by the DES were performed. In the case of random search, allocation, sequencing and batching chromosomes were generated at random and the repair algorithms (Algorithms 3 and 4) were applied if necessary.

For the optimizations without the batching chromosome, the repair algorithms were skipped, as only allocation and sequence chromosomes of uniform length are present in such cases. The results for the optimizations with batching chromosomes, were taken from the reproducibility runs from Section 5.4.

Additionally, we simulated the case study with the default scheduling heuristics of the DES. The position of an order in the global sequence is the index of the order and when there are several competing units to process an order, an order is allocated to the unit with the earliest starting time.

From the results it is obvious that the introduction of the batching decision improves the solution quality significantly, yielding average tardiness improvements of approximately 30 % compared to optimizations without batching chromosome. Random search falls behind compared to any EA-based optimization. Moreover, it can be seen that without the application of the optimizer, the DES generates a solution of clearly inferior quality. The solution provided by the DES depends on the initial sequence of orders. However, varying this sequence manually is not tractable. These results confirm that the additional effort of implementing and maintaining a sophisticated optimization procedure pays off.

It can be seen that the standard deviation between the runs is consistently larger for the runs with batching chromosome. From this it can be concluded that the batching degree of freedom strongly affects the reproducibility of the optimization. This is the price for a significantly better solution quality.

Table 5
Tardiness values in hours of the best found solutions of a set of random search runs and optimizations with and without the batching chromosome for all three cases. Tardiness values of the default solutions of the DES are also included.

	Run #	Case 1	Case 2	Case 3
Random search	1	393.44	487.32	248.19
	2	400.01	454.10	228.95
	3	379.59	443.39	169.45
	4	327.59	461.33	236.78
	5	373.13	425.01	232.04
	Average	374.75	454.23	223.08
	Std.dev.	28.45	23.00	30.86
No batching	1	304.11	347.47	111.40
	2	304.01	358.84	124.87
	3	304.01	317.80	117.42
	4	304.01	327.55	112.56
	5	304.10	352.06	104.01
	Average	304.05	340.74	114.05
	Std.dev.	0.05	17.33	7.72
With batching	1	210.13	262.37	93.26
	2	210.63	288.87	69.83
	3	261.45	307.24	98.49
	4	241.14	245.32	89.67
	5	238.54	259.44	76.01
	Average	232.38	272.65	85.45
	Std.dev.	21.95	24.93	12.07
Default		611.11	606.38	592.79

6. Conclusion and outlook

In this contribution, the medium-term scheduling of an industrial formulation plant was addressed using a Simulation–Optimization approach, with an EA as the optimizer and a commercial DES as the schedule builder to overcome the shortcomings of exact approaches with respect to the tractable level of detail of the model and the

maintainability of the solution and to improve the schedules over those provided by the heuristics that are implemented in the DES.

The novelty of this work lies in the inclusion of the batching decision and the detailed treatment of the interactions of the batching decisions with the allocation and sequencing decisions. The encoding of the batching decisions necessitated to introduce new genetic operators which can deal with variable-length chromosomes and repair algorithms that ensure a consistent set of chromosomes. As the encodings for all three degrees of freedom are largely independent, the framework is flexible and can easily be transferred to similar industrial scheduling problems.

The results demonstrate that the inclusion of the batching decision in the EA yields largely superior results compared to an optimization where only the allocation and the sequencing of the orders are optimized. In all runs of Case 1 where no batching encoding was introduced, a best tardiness value of 304 h was obtained independent of the tuning of the algorithm. With the introduction of the batching chromosome, the best parameterization of the EA yielded an average tardiness over 5 runs of approximately 222 h. Similar observations were made for Cases 2 and 3.

A low reproducibility was observed when only 1000 evaluations of schedules were performed. However, when an optimization during the night is considered, up to 4000 evaluations are possible. In this case, the standard deviation between the best tardiness solutions obtained could be reduced to acceptable values between 8.43 h and 16.67 h. The standard deviation was higher in Case 2, which is the most difficult to optimize.

Introducing the batching chromosome increases the variations between the runs. One reason for this is the gradual increase of the size of the search-space over time as more splits are introduced and which cannot be removed by the genetic operators. Since the mutation and crossover operators of the batching chromosome only add splits or keep their number constant, the number of splits never decreases throughout the optimization. If there are no changeover times between different suborders of the same order, having more splits than needed does not decrease the quality of the schedule, but increases the search space. On the other hand, introducing operators that remove splits may lead to cycles of removal and addition of splits.

Future work will include the application of the proposed approach to other case studies and investigations of the potential of model reduction and ordinal optimization approaches to mitigate the effect of the large simulation times of the DES.

CRediT authorship contribution statement

Christian Klanke: Conceptualization, Methodology, Software, Writing – original draft. **Sebastian Engell:** Conceptualization, Methodology, Writing – review & editing, Supervision, Funding acquisition.

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.

Data availability

The data that has been used is confidential.

Acknowledgements

This work was partially funded by the European Regional Development Fund (ERDF) in the context of the project OptiProd.NRW (https://www.optiprod.nrw/en). We thank Dominik Bleidorn, Engelbert Pasieka, Christian Koslowski, Vassilios Yfantis and Christian Sontag for providing comments on the manuscript.

Algorithm 1: Modified Uniform Crossover operator.

$\alpha_3 = \emptyset$ // Initialize empty allocation
for $o \in \mathcal{O}_1^s \cup \mathcal{O}_2^s$
  if $o \in \mathcal{O}_1^s \wedge o \in \mathcal{O}_2^s$ **then**
    $u = \left\{ \begin{array}{l} \alpha_1[o], \textit{if } \mathbb{U}(0,1) < 0.5 \\ \alpha_2[o], \textit{otherwise} \end{array} \right\}$
    $\alpha_3 \leftarrow \alpha_3 \cup \{o, u\}$
  else if $o \in \mathcal{O}_1^s$ **then**
    $\alpha_3 \leftarrow \alpha_3 \cup \{o, \alpha_1[o]\}$
  else
    $\alpha_3 \leftarrow \alpha_3 \cup \{o, \alpha_2[o]\}$
  endif
endfor

Algorithm 2: Modified Cycle Crossover.

$\pi_3 = \emptyset$ // Initialize empty sequence
$allPos \leftarrow \{1, 2, \ldots, |\pi_1|\}$
$pos_{P1} \overset{R}{\leftarrow} allPos$
$allPos \leftarrow allPos \setminus pos_{P1}$
$\pi_3[pos_{P1}] \leftarrow \pi_1[pos_{P1}]$
$pos_{P2} \leftarrow pos_{P1}$
$exitFirst \leftarrow false$
// First round, cycle sequences
while !exitFirst
  $pos_{P2} \leftarrow min(pos_{P2}, |\pi_2|)$
  if $\exists\, pos_{P1} \leftarrow \pi_1.getIndex(\pi_2[pos_{P2}])$ **then**
    // Value exists in $P1$
    if $\exists\, \pi_3[pos_{P1}]$ **then**
      // Position taken, first round ends
      $exitFirst \leftarrow true$
    else
      $\pi_3[pos_{P1}] \leftarrow \pi_1[pos_{P1}]$
      $pos_{P2} \leftarrow pos_{P1}$
    endif
  else
    // Value doesn't exist in $P1$
    $pos_{P1} \leftarrow \left\{ \begin{array}{l} pos_{P2} - 1, \textit{if } pos_{P2} > |\pi_1| \\ pos_{P2} + 1, \textit{otherwise} \end{array} \right\}$
    if $\exists\, \pi_3[pos_{P1}]$ **then**
      $exitFirst \leftarrow true$
    else
      $\pi_3[pos_{P1}] \leftarrow \pi_2[pos_{P2}]$
    endif
    $pos_{P2} \overset{R}{\leftarrow} allPos$
    $allPos \leftarrow allPos \setminus pos_{P2}$
  endif
endwhile
// Merge missing elements from $P1$ and $P2$
$\pi_{miss} \leftarrow zip(\pi_1 \setminus \pi_3, \pi_2 \setminus \pi_3)$
// Second round, fill missing values in order
for $x \in \pi_{miss}$
  $i \leftarrow getFirstEmptyPos(\pi_3)$
  $\pi_3[i] \leftarrow x$
endfor

Appendix

See Algorithms 1–7.

Algorithm 3: Repair algorithm - Sequence.

```
missing = getMissing(β, π)
if missing != ∅ then
    for o ∈ missing
        idx = π.getIndex(o)
        π.insert(idx, o)
        remOrders = remOrders ∪ o
    endfor
    remOrders = remOrders ∪ getRedundantElements(π, β)
    π = π \ remOrders
endif
```

Algorithm 4: Repair algorithm - Allocation.

```
missing = getMissing(β, α)
redundant = getredundant(β, α)
if missing != ∅ then
    for o ∈ missing
        u = getRandomUnit()
        α = α ∪ {o, u}
    endfor
endif
if redundant != ∅ then
    for o ∈ redundant
        α = α \ {o, α[o]}
    endfor
endif
```

Algorithm 5: Batching crossover operator.

```
β3 = β1 ∩ β2                          // Keep shared production orders
for {o, splits} ∈ β1
    if x ← 𝕌(0, 1)< 0.5 then
        β3 ← β3 ∪ {o, splits}
    endif
endfor
for {o, splits} ∈ β2
    if x ← 𝕌(0, 1)< 0.5 then
        β3 ← β3 ∪ {o, splits}
    endif
endfor
```

Algorithm 6: *AddRandomSplit* and *AddRandomSplitWeighted* mutation operator.

```
case Batching Mutation Operator
    AddRandomSplit: o ← 𝒪[𝕌(0, |𝒪| − 1)]
    AddRandomSplitWeighted:
        o ← {o' ∈ 𝒪 | p(o') = T_o',hist / Σ_o'∈𝒪 T_o',hist}
endcase
if o ∈ β.getOrders() then
    splits ← β[o]
    splits ← distributeUniformly(|splits| + 1)
    β ← β ∪ {o, splits}
else
    splits ← {o, distributeUniformly(2)}
    β ← β ∪ {o, splits}
endif
```

Algorithm 7: *ChangeExistingSplit* mutation operator.

```
amt ← 0.1                             // Discrete split amount
if β ≠ ∅ then
    splits ←R β.getSplits()
                                      // Find split amount to increase
    a_inc ← {a | a ∈ splits ∧ a ≤ 1 - amt}
    splits ← splits \ a_inc
                                      // Find split amount to decrease
    a_dec ← {a | a ∈ splits ∧ a ≥ amt}
                                      // Increase and decrease splits
    a_inc ← a_inc + amt
    splits ← splits ∪ a_inc
    a_dec ← a_dec − amt
endif
```

References

Ackermann, S., Fumero, Y., & Montagna, J. M. (2022). Taking advantage of order consolidation in simultaneous batching and scheduling of multiproduct batch plants. *Computers and Chemical Engineering*, *156*, Article 107564.

Allal, A., Sahnoun, M., Adjoudj, R., Benslimane, S. M., & Mazar, M. (2021). Multi-agent based simulation-optimization of maintenance routing in offshore wind farms. *Computers & Industrial Engineering*, *157*(January 2020), Article 107342.

Amaran, S., Sahinidis, N. V., Sharda, B., & Bury, S. J. (2016). Simulation optimization: a review of algorithms and applications. *Annals of Operations Research*, *240*(1), 351–380.

Awad, M., Mulrennan, K., Donovan, J., Macpherson, R., & Tormey, D. (2022). A constraint programming model for makespan minimisation in batch manufacturing pharmaceutical facilities. *Computers and Chemical Engineering*, *156*, Article 107565.

Azzaro-Pantel, C., Bernal-Haro, L., Baudet, P., Domenech, S., & Pibouleau, L. (1998). A two-stage methodology for short-term batch plant scheduling: discrete-event simulation and genetic algorithm. *Computers and Chemical Engineering*, *22*(10), 1461–1481.

Basán, N. P., Cóccola, M. E., García del Valle, A., & Méndez, C. A. (2020). Scheduling of flexible manufacturing plants with redesign options: A MILP-based decomposition algorithm and case studies. *Computers and Chemical Engineering*, *136*, Article 106777.

Beyer, H.-G., & Schwefel, H.-P. (2002). Evolution strategies - A comprehensive introduction. *Natural Computing*, *1*, 3–52.

Castro, P. M., Erdirik-Dogan, M., & Grossmann, I. E. (2008). Simultaneous batching and scheduling of single stage batch plants with parallel units. *AIChE Journal*, *54*(1), 183–193.

Chiang, T.-c., & Lin, H.-j. (2013). A simple and effective evolutionary algorithm for multiobjective flexible job shop scheduling. *International Journal of Production Economics*, *141*(1), 87–98.

Defersha, F. M., & Chen, M. (2012). Jobshop lot streaming with routing flexibility, sequence-dependent setups, machine release dates and lag time. *International Journal of Productions Research*, *50*(8), 2331–2352.

Defersha, F. M., & Rooyani, D. (2020). An efficient two-stage genetic algorithm for a flexible job-shop scheduling problem with sequence dependent attached/detached setup, machine release date and lag-time. *Computers & Industrial Engineering*, *147*(June), Article 106605.

Dong, S., & Medeiros, D. J. (2012). Minimising schedule cost via simulation optimisation: An application in pipe manufacturing. *International Journal of Productions Research*, *50*(3), 831–841.

Eiben, A. E., & Smith, J. E. (2004). *Introduction to evolutionary computing*. Springer-Verlag Berlin Heidelberg.

Escobet, T., Puig, V., Quevedo, J., Palá-Schönwälder, P., Romera, J., & Adelman, W. (2019). Optimal batch scheduling of a multiproduct dairy process using a combined optimization/constraint programming approach. *Computers and Chemical Engineering*, *124*, 228–237.

Figueira, G., & Almada-Lobo, B. (2014). Simulation Modelling Practice and Theory Hybrid simulation – optimization methods : A taxonomy and discussion. *Simulation Modelling Practice and Theory*, *46*, 118–134.

Fumero, Y., Corsano, G., & Montagna, J. M. (2012). Planning and scheduling of multistage multiproduct batch plants operating under production campaigns. *Annals of Operations Research*, *199*(1), 249–268.

Georgiadis, G. P., Elekidis, A. P., & Georgiadis, M. C. (2019). Optimization-based scheduling for the process industries: From theory to real-life industrial applications. *Processes*, *7*(7), 438.

Harjunkoski, I., Maravelias, C. T., Bongers, P., Castro, P. M., Engell, S., Grossmann, I. E., Hooker, J., Méndez, C., Sand, G., & Wassick, J. (2014). Scope for industrial applications of production scheduling models and solution methods. *Computers and Chemical Engineering*, *62*, 161–193.

He, Y., & Hui, C. W. (2007). Genetic algorithm for large-size multi-stage batch plant scheduling. *Chemical Engineering Science*, *62*(5), 1504–1523.

Honkomp, S. J., Lombardo, S., Rosen, O., & Pekny, J. F. (2000). The curse of reality - Why process scheduling optimization problems are difficult in practice. *Computers and Chemical Engineering*, *24*(2–7), 323–328.

Ishikawa, S., Kubota, R., & Horio, K. (2015). Effective hierarchical optimization by a hierarchical multi-space competitive genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, *42*(24), 9434–9440.

Istokovic, D., Perinic, M., Vlatkovic, M., & Brezocnik, M. (2020). Minimizing total production cost in a hybrid flow shop: a simulation-optimization approach. *International Journal of Simulation Modelling*, *19*(4), 559–570.

Jankauskas, K., Papageorgiou, L. G., & Farid, S. S. (2019). Fast genetic algorithm approaches to solving discrete-time mixed integer linear programming problems of capacity planning and scheduling of biopharmaceutical manufacture. *Computers and Chemical Engineering*, *121*, 212–223.

Kemp, D. (2003). Discrete-event simulation: Modeling, programming, and analysis. *Journal of the Royal Statistical Society: Series D (the Statistician)*, *52*(3), 408–409.

Kim, H.-J., & Lee, J.-H. (2021). Scheduling uniform parallel dedicated machines with job splitting, sequence-dependent setup times, and multiple servers. *Computers & Operations Research*, *126*, Article 105115.

Kim, Y. H., & Su, K. R. (2020). Insertion of new idle time for unrelated parallel machine scheduling with job splitting and machine breakdowns. *Computers & Industrial Engineering*, *147*(3), Article 106630.

Klanke, C., Bleidorn, D., Koslowski, C., Sonntag, C., & Engell, S. (2021). Simulation-based scheduling of a large-scale industrial formulation plant using a heuristics-assisted genetic algorithm. In *GECCO '21: Proceedings of the genetic and evolutionary computation conference companion* (pp. 1587–1595). Association for Computing Machinery.

Klanke, C., Bleidorn, D. R., Yfantis, V., & Engell, S. (2021). Combining constraint programming and temporal decomposition approaches - Scheduling of an industrial formulation plant. *Lecture Notes in Computer Science*, *12735*, 133–148.

Klanke, C., Yfantis, V., Corominas, F., & Engell, S. (2020). Scheduling of a large-scale industrial Make-and- Pack process with finite intermediate buffer using discrete-time and precedence-based models. *Computer Aided Chemical Engineering*, *47*, 1153–1158.

Larranaga, P., Kuijpers, C. M. H., Murga, R. H., & Yurramendi, Y. (1996). Learning Bayesian network structures by searching for the best ordering with genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics Part A: Systems and Humans*, *26*(4), 487–493.

Lee, J.-H., & Kim, H.-J. (2021). A heuristic algorithm for identical parallel machine scheduling: splitting jobs, sequence-dependent setup times, and limited setup operators. *Flexible Services and Manufacturing Journal*, *33*(4), 992–1026.

Lee, H., & Maravelias, C. T. (2017). Mixed-integer programming models for simultaneous batching and scheduling in multipurpose batch plants. *Computers and Chemical Engineering*, *106*, 621–644.

Lin, J. T., & Chen, C. M. (2015). Simulation optimization approach for hybrid flow shop scheduling problem in semiconductor back-end manufacturing. *Simulation Modelling Practice and Theory*, *51*, 100–114.

Maravelias, C. T. (2012). General framework and modeling approach class ification for chemical production scheduling. *AIChE Journal*, *59*(4), 215–228.

Méndez, C. A., Cerdá, J., Grossmann, I. E., Harjunkoski, I., & Fahl, M. (2006). State-of-the-art review of optimization methods for short-term scheduling of batch processes. *Computers and Chemical Engineering*, *30*(6–7), 913–946.

Moser, M., Herrmann, M., & Engell, S. (1992). Avoiding scheduling errors by partial simulation of the future. In *Proceedings of the 31st IEEE conference on decision and control* (pp. 411–412).

Negahban, A., & Smith, J. S. (2014). Simulation for manufacturing system design and operation: Literature review and analysis. *Journal of Manufacturing Systems*, *33*(2), 241–261.

Novas, J. M. (2019). Production scheduling and lot streaming at flexible job-shops environments using constraint programming. *Computers & Industrial Engineering*, *136*(July), 252–264.

Oyarbide-Zubillaga, A., Goti, A., & Sanchez, A. (2008). Preventive maintenance optimisation of multi-equipment manufacturing systems by combining discrete event simulation and multi-objective evolutionary algorithms. *Production Planning and Control*, *19*(4), 342–355.

Papageorgiou, L. G., & Pantelides, C. C. (1993). A hierarchical approach for campaign planning of multipurpose batch plants. *Computers and Chemical Engineering*, *17*, S27–S32.

Piana, S., & Engell, S. (2010). Hybrid evolutionary optimization of the operation of pipeless plants. *Journal of Heuristics*, *16*(3), 311–336.

Potts, C. N., & Kovalyov, M. Y. (2000). Scheduling with batching: a review. *European Journal of Operational Research*, *120*(2), 228–249.

Potts, C., & van Wassenhove, L. (1992). Integrating scheduling with batching and lot-sizing a review of algorithms and complexity. *Journal of the Operational Research Society*, *43*(5), 395–406.

Prasad, P., & Maravelias, C. T. (2008). Batch selection, assignment and sequencing in multi-stage multi-product processes. *Computers and Chemical Engineering*, *32*(6), 1106–1119.

Razali, N. M., & Geraghty, J. (2011). Genetic algorithm performance with different selection strategies in solving TSP. *Proceedings of the World Congress on Engineering 2011, WCE 2011*, *2*, 1134–1139.

Reeves, C. R. (1995). A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, *22*(1), 5–13.

Roe, B., Papageorgiou, L. G., & Shah, N. (2005). A hybrid MILP/CLP algorithm for multipurpose batch process scheduling. *Computers and Chemical Engineering*, *29*(6 SPEC. ISS.), 1277–1291.

Schwindt, C., & Trautmann, N. (2000). Batch scheduling in process industries: An application of resource-constrained project scheduling. *OR Spektrum*, *22*(4), 501–524.

Shah, N., & Pantelides, C. C. (1991). Optimal long-term campaign planning and design of batch operations. *Industrial and Engineering Chemistry Research*, *30*(10), 2308–2321.

Smith, J. S. (2003). Survey on the use of simulation for manufacturing system design and operation. *Journal of Manufacturing Systems*, *22*(2), 157–171.

Sundaramoorthy, A., & Maravelias, C. T. (2008). Modeling of storage in batching and scheduling of multistage processes. *Industrial and Engineering Chemistry Research*, *47*(17), 6648–6660.

Sundaramoorthy, A., & Maravelias, C. T. (2008). Simultaneous batching and scheduling in multi-stage processes with storage constraints. *AIChE Annual Meeting, Conference Proceedings*, 1546–1555.

Syberfeldt, A., & Lidberg, S. (2012). Real-world simulation-based manufacturing optimization using Cuckoo Search. In *Proceedings - Winter simulation conference*. IEEE.

Tasoglu, G., & Yildiz, G. (2019). Simulated annealing based simulation optimization method for solving integrated berth allocation and quay crane scheduling problems. *Simulation Modelling Practice and Theory*, *97*, Article 101948.

Urlings, T., Ruiz, R., & Serifoglu, F. S. (2010). Genetic algorithms with different representation schemes for complex hybrid flexible flow line problems. *International Journal of Metaheuristics*, *1*(1), 30.

Voß, S., & Witt, A. (2003). Batching in der Produktionsplanung – Projektplanung mit reihenfolgeabhängigen Rüstkosten. *Zeitschrift Für Planung*, *14*(1), 75–89.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82.

Wongthatsanekorn, W., & Phruksaphanrat, B. (2015). Genetic algorithm for short-term scheduling of make-and-pack batch production process. *Chinese Journal of Chemical Engineering*, *23*(9), 1475–1483.

Yang, T., Kuo, Y., & Cho, C. (2007). A genetic algorithms simulation approach for the multi-attribute combinatorial dispatching decision problem. *European Journal of Operational Research*, *176*(3), 1859–1873.

Yazdani, M., Kabirifar, K., Fathollahi-Fard, A. M., & Mojtahedi, M. (2021). Production scheduling of off-site prefabricated construction components considering sequence dependent due dates. *Environmental Science and Pollution Research*.

Yfantis, V., Babskiy, A., Klanke, C., Ruskowksi, M., & Engell, S. (2022). An improved optimization model for scheduling of an industrial formulation plant based on integer linear programming. In Y. Yamashita, & M. Kano (Eds.), *Computer Aided Chemical Engineering*: *vol. 49*, *14th International symposium on process systems engineering* (pp. 487–492). Elsevier.

Yfantis, V., Siwczyk, T., Lampe, M., Kloye, N., Remelhe, M., & Engell, S. (2019). Iterative medium-term production scheduling of an industrial formulation plant. *Computer Aided Chemical Engineering*, *46*, 19–24.