

Evolutionary Algorithm Based Real-time Scheduling via Simulation-Optimization for Multiproduct Batch Plants

Engelbert Pasieka^{a*}, Sebastian Engell^b

^a INOSIM Software GmbH, Joseph-von-Fraunhofer-Str. 20, 44227 Dortmund, Germany

^b TU Dortmund, Department of Biochemical and Chemical Engineering, Emil-Figge-Str. 70, 44221 Dortmund, Germany

* engelbert.pasieka@inosim.com

ABSTRACT

Production scheduling in the process industry is an area of significant activity in research and of great practical importance for the performance of industrial companies. In the vast majority of research papers, the scheduling problem is formulated as an off-line problem where a number of jobs is scheduled on a number of resources and the efficiency of the formulation and the solution algorithms is discussed. In reality, however, scheduling is a continuous activity that has to react to the arrival of new orders, to variations in processing times, breakdowns, lack of resources etc. This is termed real-time (or online) scheduling. Available commercial solutions usually provide solutions with relatively long update intervals due to the necessary computation times and a delayed flow of information from the manufacturing execution systems where the data on the current state of the production is collected. Thus the computed schedules are outdated quickly, if not already at the point in time when they become available, and new orders and all kinds of variations and disturbances are handled by the operators and plant supervisors. There has been little concern for systematic solutions for real-time scheduling that continuously react to the changing situation in the plant. In this contribution we present a simulation-optimization approach to real-time scheduling where a detailed discrete-event simulation of the plant is coupled with an evolutionary algorithm. The simulation model is continuously updated with the latest plant data, and the optimization algorithm continuously improves the schedule based on the currently available information. We validate our approach using an example of a multiproduct, multistage batch plant in the pharmaceutical industry from the literature, demonstrating that the proposed approach can generate high-quality solutions quickly. We compare the results with those of an ideal (clairvoyant) scheduler, demonstrating that our method effectively manages unforeseen disturbances.

Keywords: Modelling and Simulation, Large Scale Design, Planning/Scheduling.

INTRODUCTION

Production scheduling is a decision-making process that determines the optimal allocation to resources and sequence of operations in production processes. This improves plant efficiency, reduces operational costs, and improves customer satisfaction by meeting the promised delivery dates. Scheduling in the scientific literature is typically considered in an offline setting where a static problem for a given production environment, given jobs and timing constraints, and a given cost function is solved. In industrial practice, scheduling is usually done periodically with relatively long intervals between the

computations or manual generations of updated schedules due to the complexity and required computation time as well as the lack of fast feedback from the shop floor.

Real-world production environments however are subject to frequent unforeseen events such as machine breakdowns, new order arrivals, varying processing times and changes of the availability of resources. Such events may render the available schedules inefficient or even infeasible so that adaptations are needed.

Real-time scheduling systems have the task to generate updates of the production plans based on a continuous exchange of information with the shop floor systems. These plans should be executable by the real plant

because otherwise time-consuming modifications or refinements are necessary. So there are two, contradictory requirements: The computation of the schedules must be fast, where sufficiently fast is defined by the execution times of the production plant. When an operation is finished, the next executable operation must be started with a delay that should not be larger than a small fraction of the average execution time. On the other hand, for a schedule to be executable, the scheduling system should be based on a detailed model of the plant that includes all relevant conditions and constraints for the execution of the production plans. This excludes the use of simplified models which consider only basic constraints as e.g. the sequence of operations within jobs, exclusivity of the use of resources and changeover times. Many existing approaches fall short to deliver timely responses and to generate schedules that are interpretable and executable by plant operators [1]. One possible way of solving the real-time or online scheduling problem is to see the scheduler as a controller that continuously monitors the current state and applies corrective actions to keep the system on track toward the reference trajectory [2]. Then offline scheduling is analogous to providing the reference trajectory or setpoint of a dynamic system. However, while this is a convincing analogy, there are fundamental differences, in particular regarding what “tracking” means for schedules. Our approach here therefore is similar to the “all in one” optimizing control approach [3]. Specifically, we employ simulation-optimization to realize this idea.

In [4], MILP-based methods using state-space formulations were introduced for online scheduling in a feedback loop. However, as noted in [4], these approaches require advanced solution techniques to handle high-complexity and high-fidelity problems to maintain computational feasibility.

Simulation-optimization (SO) is well suited for this task as high-fidelity simulation models that represent the production process in detail can be used and therefore the scheduler will generate production plans that the plant operators can implement without modification. Simulation models can easily be adapted to changes of the structure (e.g. available resources) and the parameters (e.g. predicted execution times) of the production system and to new jobs and re-computed schedules. For scheduling on the basis of detailed complex stimulation models, rigorous scheduling methods usually are not well suited because the transformation of the simulation model into the underlying formalism leads to very complex models and excessive computation times. Therefore in SO generally metaheuristics as e.g. evolutionary algorithms are applied. Such algorithms are less performing than rigorous algorithms for the solution of MILPs or constraint satisfaction problems as long as the solution times of the

rigorous algorithms are acceptable, but have the advantage of being more generally applicable and leading to acceptable solutions early in the solution process, however without guarantees for the closeness of the solution to the true optimum. In [5], a successful application of SO with evolutionary algorithms to a complex real-world industrial scheduling problem was reported. It was also shown in [6] for general video game playing (GVGA) that evolutionary algorithms (EA) with simulation models compete well in environments where changes of the decision space are frequent and fast responses are crucial.

Our proposed simulation-optimization approach provides greater flexibility and adaptability for handling complex scheduling problem and the results are easier to interpret, thereby facilitating practical adoption in industrial settings. In [7], we presented the idea of a reactive real-time scheduling approach based on SO. In this paper, we extend our previous work by providing a more detailed explanation of the interaction between the real plant and the scheduling system, by exploring the influence of critical parameters and by discussion a case study of a complex pharmaceutical batch production [8], simulating a typical 8-hour production shift with major disruptions.

APPROACH

Our online scheduling framework uses simulation-optimization with real-time feedback from the production process that is used to update the simulation model to align with the state of the production. The framework contains two components, the scheduling system and the production system. Figure 1 shows the components and their interactions.

Scheduling System

The evolutionary algorithm runs continuously and improves the schedules gradually based on the currently available information. The simulation system (a detailed commercial discrete-event simulator) uses heuristics and logic constraints to generate the detailed schedules, i.e. it acts as the schedule builder. The model in the scheduler is updated continuously with the latest information from the plant. The detailed schedule that is passed to the production management system is executable on the shop floor. This is achieved by the execution logic and, if needed repair mechanisms that are implemented in the simulator. E.g., no new operation can be started on a resource which has become unavailable. Transferring schedules that are generated fast after changes in the plant or in the structure of the orders provide an immediate response to disturbances. As this happens after few iterations of the EA, their quality will be lower, but in the following iterations of the EA they are further improved.

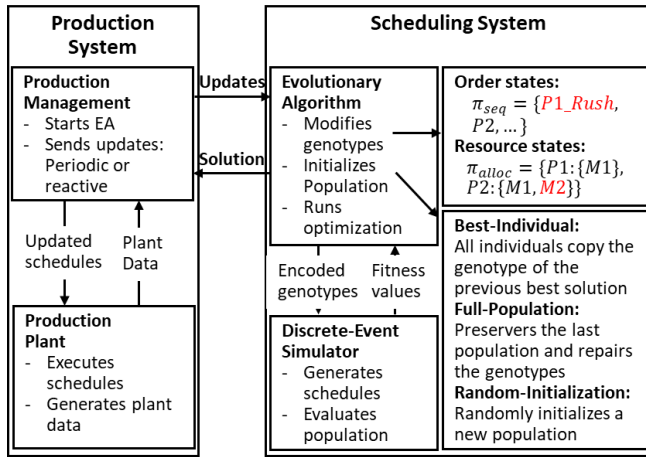


Figure 1. The reactive real-time scheduling framework integrates a scheduling system with an evolutionary algorithm and discrete event simulator and a production system with data access to the plant to dynamically adapt to changes and disruptions in the plant. The scheduling system modifies genotypes representing order sequences and resource allocations and uses a discrete-event simulation to evaluate schedules. The evolutionary algorithm iteratively updates and refines schedules for the current model of the production system. The production system provides data from the plant and triggers updates in response to events such as delays, rush orders, and maintenance tasks, ensuring continuous alignment between schedules and the current plant state.

In the case investigated here, the scheduling solutions are encoded as genotypes which contain a global sequence of orders, e.g. $\pi_{seq} = \{P1, P2, P3, \dots\}$, and the allocation of orders to resources, e.g., $\pi_{alloc} = \{P1:\{M1\}, P2:\{M1, M2\}, \dots\}$. In more complex examples, further degrees of freedom can be encoded, e.g. the splitting of customer orders into production jobs (see [4]). The EA explores the space of feasible schedules through selection, recombination, and mutation. The repeated execution of the EA starts with the modification (if needed) and initialization of the genotypes. In each iteration, the algorithm evaluates the fitness of each genome in the population by the execution on the updated simulation model, selects the most promising candidates, and applies recombination and mutation to generate new solutions. A survivor selection process ensures which solutions are carried forward. During the evaluation of the genomes, the simulator builds a detailed schedule that complies with all known constraints. The best available detailed schedule is transferred to the plant when an update is requested.

Production System

The production system provides real-time information to the scheduling system. It consists of the pro-

duction management system and the production plant itself. The production management system initiates and parametrizes the evolutionary algorithm and monitors the real-time data from the production plant for deviations that require updates of the scheduling model or of the genotypes and of the genomes of the solutions in the population. Updates are performed either periodically, updating the current state of the execution of the schedule and parameters as e.g. expected execution times, or reactively in response to significant disruptive events, such as rush orders, breakdowns, large delays in the execution of operations or maintenance requirements.

Information Exchange

Figure 2 illustrates the information exchange between the production system and the scheduling system.

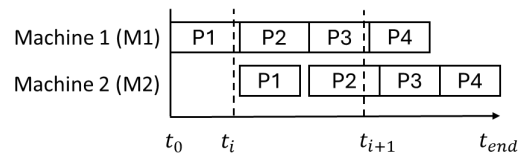


Figure 2. Gantt Chart with update events. M1 and M2 process the orders sequentially (first on M1, then on M2).

At each update instance t_i , the production system provides the scheduling system with real-time operational data, including the current status of ongoing tasks, estimated remaining processing times, and details of disruptions such as machine breakdowns, maintenance tasks, or urgent new orders. At the same time, the scheduling system sends an updated detailed schedule for the next period starting at t_{i+1} .

After the exchange at t_i , the scheduling system updates its simulation model and configures the evolutionary algorithm by adjusting decision variables if needed. It then continues optimizing the schedule for the next period (starting at t_{i+1}). Operations that are already in progress (e.g., P2 on M1) remain fixed.

Disruptive events, such as maintenance activities or new order arrivals, trigger immediate reactive updates. The scheduling system instantly integrates this new information, modifies the simulation model and reconfigures the evolutionary algorithm. After a predefined interval after such disruptive events, a revised schedule is generated and sent to the production system after which the evolutionary algorithm continues its optimization for the rest of the regular update interval.

Reinitialization of the population

A key element in designing the scheduling system is how the population is managed between updates, as this can be used to preserve information between updates and to control the balance between exploiting current solutions and exploring new ones. We investigated three options:

Best-Individual: The best individual from the previous population (i.e., the best schedule found before t_i) is copied into the new population, and only the parts affected by the changed conditions are adjusted randomly.

Full-Population: The entire population is carried over to the next generation, and the solutions are only repaired to fit the updated constraints (e.g., new tasks or changed machine availability). The balance of exploration and exploitation largely depends on the diversity of the previous population.

Random-Initialization: The entire population is reinitialized randomly, producing a new and diverse population. This enhances the exploration and may help avoid local minima that arise when solutions become too similar.

The three strategies can be mixed, e.g. by updating a fraction of the population with random solutions after a disruptive event. Another option is to modify the strategy parameters of the EA, e.g. the mutation strength, at disruptive events and then including them in the optimization process (Co-evolution, [9]).

EXPERIMENTS

The case study addressed in this work is taken from Kopanos, et al., [8]. It is a multiproduct batch plant with 17 units (machines) that are organized in 6 stages (Figure 3). The problem, which is a variant of a hybrid flow shop problem, comprises 12 instances that vary in the number of orders, the objectives, and in the storage policy. The features of the problem include limited product-unit flexibility, machine-dependent processing times, sequence-dependent changeover times, and product-specific recipes, meaning that certain jobs are not processed on some of the available stages.

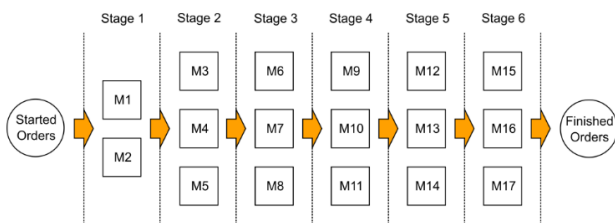


Figure 3: Plant layout of the case study.

A series of experiments were executed to evaluate the performance of the proposed real-time scheduling system and to show the effects of the length of the update intervals, the time required to evaluate one generation, and of the initialization strategy. The experiments simulate one production shift in the multiproduct batch plant and include disruptive events. The results are compared to the outcomes from using a clairvoyant scheduler that anticipates all disruptive events. This solution was computed by optimizing the full schedule over the full

time horizon with the disruptive events taken into account. The same evolutionary algorithms was used and run over 1000 generations.

In total, 20 combinations of parameters were evaluated, with each case being repeated 20 times to account for the inherent randomness of the evolutionary algorithm. The experiments differ in the length of the update intervals of the production system, the time before the immediate response must be provided, the time to evaluate one generation of the scheduling system, and the initialization strategy after update events.

Fixed Parameters

The following parameters were used across all experiments to ensure comparability:

Problem setting: Initially 30 orders have to be fulfilled within a time horizon of 32 hours. For each order there is a due date. All orders are released at time 0. The prime objective (Equation (1)) is the minimization of the maximum tardiness T . If the tardiness is below 0, the objective shifts to the largest negative earliness. This pushes the completion times C_i of the orders i further away from their due dates d_i if all due dates can be met.

$$F = \max_i (C_i - d_i) \quad (1)$$

Simulation Horizon: Each simulation run consists of 8 hours (28,800 seconds) of production, representing a typical production shift. Note that this is only a fraction of the execution time of a schedule. The schedules are evaluated over the full execution time, i.e. using the completion times of the orders that finish in the simulation horizon and the predicted (simulated) completion times of the remaining orders.

Initial Solution: The simulation was initialized with a high-quality scheduling solution that was computed in a separate run with the same objective and 1000 generations, serving as a reference schedule. The initial solution has an objective value of -1.34 (i.e. all orders finish early). The total time to complete the full production schedule is 30 hours and 23 minutes.

Disruptive Events: Table 1 lists the eight disruptive events that were introduced during each simulation to represent real-world disturbances.

Table 1: Disruptive events and their arrival times.

Event	Time (sec)	Info
Release of P01_Rush	2400	Due in 12h
Release of P03_Rush	4200	Due in 12h
Maintenance of M1	4800	1h duration
Maintenance of M3	7200	1h duration
Maintenance of M19	12000	1h duration
Release of P05_Rush	13800	Due in 12h
Maintenance of M13	18000	1h duration
Maintenance of M16	21600	0.5h duration

All maintenance orders were non-preemptive,

meaning they had to be executed after the current operation on the respective machine had completed.

Population Size: The population consists of 32 individuals, which are evaluated in parallel using 32 threads on a machine with 16 cores (Intel Core i9-14900K). The offspring size was set to 32, with an elitist size of 16.

Mutation Operator: We used a permutation mutation, which selects a subsection of the gene and shuffles it randomly. The mutation rate was set to 1, meaning every individual underwent mutation.

Crossover Operator: The cycle crossover operator was applied to preserve as much information as possible about the absolute position of the elements [10]. The crossover rate was set to 1.

Parent Selection: Rank-based selection was used to choose parent individuals for the next generation.

Survivor Selection: Random survivor selection was employed, with an elitist fraction of 50%. The top 50% of the individuals from the previous generation were retained, while the remaining individuals were selected randomly from the remaining population.

Decision Variables of the EA: Only the sequence of orders, e.g., $\pi_{seq} = \{P1, P2, P3, \dots\}$, was encoded in the genomes of the individuals of the EA. The allocation of orders to machines was performed by the simulator using a greedy FIFO rule. Each operation is executed on the machine where it finishes the earliest.

Variable Parameters

To study their effect on the performance of the scheduling system, the following parameters were varied:

Update Interval: The length of the periodic update interval was set to 60 and to 300 seconds.

Simulation time: This parameter is a fictitious time for the evaluation of one generation of solutions. It determines the number of generations that are evaluated. Simulations times of 3s and 12s were tested.

Immediate Response Time: This is the time when after a disruptive event a new solution is sent to the production system. The minimum time is the time needed to evaluate on generation of solutions. The second value that was tested is 60 seconds.

Initialization: The modes *Full-Population*, *Best-Individual* and *Random* were tested.

RESULTS

Figure 4 shows the distribution of the mean fitness values for the three different initialization modes. The mean fitness values for all experiments at the minimum time needed to evaluate the generation of solutions after each disruptive event in Table 1.

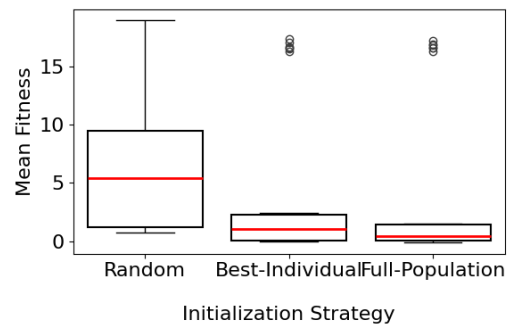


Figure 4: Distribution of the schedule quality for different initialization strategies after the disruptive events.

The Random initialization is clearly inferior. There are only small differences between the *Best-Individual* and *Full-Population* strategies. As the latter performs slightly better, only results for this strategy are shown in the sequel.

Figure 5 shows the evolution of the mean fitness over all 20 runs and of the best 25% of the runs at the end of the simulation horizon over the simulated 8 hours of production with the disruptive events listed in Table 1 for different configurations (300 s periodic update interval, 3 and 12 s/gen, immediate response after one generation (0) and after 60 seconds). Additionally, the performance without replanning is indicated by the solid black line.

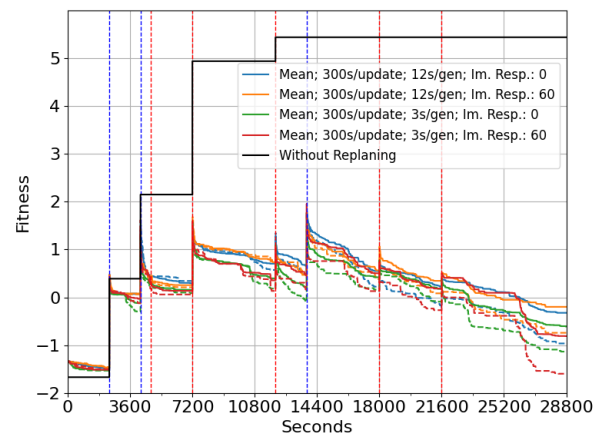


Figure 5: Mean (colored solid line) and mean of best 25% (colored dashed line) fitness values of 20 experimental runs for different configurations (300 s/update, 3 and 12 s/gen, im. Resp. after one generation and 60 seconds). The disruptive events are marked as dashed vertical lines (blue for maintenances and red for new order arrivals).

Table 2 shows the results for all configurations. The clairvoyant scheduler uses the same algorithm but has the a priori information about all disruptive events. This provides a lower bound that cannot be achieved by a real-time scheduler.

Table 2: Best, mean, mean of best 25% of fitness values for the different configurations (formatted as “seconds per update, seconds per generation, immediate response time”), for the clairvoyant scheduler, and for the case without replanning.

Configurations	Best	Mean	Mean Best 25%
300, 12, 60	-1.30	-0.20	-0.74
300, 12, 0	-1.67	-0.32	-0.96
300, 3, 60	-1.76	-0.81	-1.60
300, 3, 0	-2.10	-0.61	-1.14
60, 12, 0	-1.68	-0.52	-1.08
60, 12, 60	-1.61	-0.23	-0.74
60, 3, 0	-2.10	-0.68	-1.34
60, 3, 60	-1.53	-0.24	-0.83
Clairvoyant	-4.37		
W/o replan	5.44		

The differences between the configurations are relatively small when the results at the end of the simulation horizon are compared. The best configuration realizes 75 – 80% of the possible improvement. The simulation time has a negative influence but due to the long optimization interval at the end it is not very pronounced. Between disruptive events the shorter simulation time leads to significantly faster improvements. The influence of the immediate response time is small and nonuniform.

For the disruptive events considered here, the deterioration of the performance of the production system with rescheduling is relatively small, pointing to the system not being operated under full load.

SUMMARY

We proposed an approach to real-time scheduling which is based on simulation-optimization using a detailed discrete-event simulator with schedule building heuristics and an evolutionary algorithm that handles the key decision variables. The main idea is to run the evolutionary algorithm continuously with updates of the simulation model and of the decision vector being performed periodically and in response to disruptive events. Maintaining the previous population except for changes required by changes of the decision parameters is preferable over other possible initializations. The resulting performance is good even for low numbers of generations that are evaluated during the update interval.

We characterized the performance by comparison with the execution of the schedule without changes due to the disruptive events and with the results obtained with a fictitious clairvoyant scheduler which cannot be achieved in a situation where these events are not known before.

We considered the influence of simulation time on the performance. Other cases with demanding simulation times may require additional simplifications or algorithmic

optimizations to meet the time constraints. This aspect will be further studied.

In this paper we only reported results for a small number of relatively infrequent disruptive events for lack of space. It will be studied in the future which influence the parameters of the algorithm have in situations where the plant operates at the capacity limit.

REFERENCES

1. Espinaco F, Henning P. Industrial Rescheduling Approaches: Where Are We and What is Missing? *Production Research–Americas*, pp 461–467 (2023).
2. Engell S. Uncertainty, decomposition and feedback in batch production scheduling. *Proc. ESCAPE19*, pp 43–62 (2009).
3. Engell S. Feedback control for optimal process operation. *Journal of Process Control*, Vol. 17, pp 203–219 (2007).
4. Gupta D, Maravelias CT. A General State-Space Formulation for Online Scheduling. *Processes*, vol. 5, no. 4, pp 69 (2017).
5. Klanke C, Engell S. Scheduling and batching with evolutionary algorithms in simulation-optimization of an industrial formulation plant. *Computers & Industrial Engineering*, Vol. 174 (2022).
6. Gaina RD, Devlin S, Lucas SM, Perez-Liebana, D. Rolling Horizon Evolutionary Algorithms for General Video Game Playing. *IEEE Conference on Computational Intelligence and Games* (2020).
7. Pasięka E, Engell S. Reactive Real-time Scheduling Using Simulation-Optimization and Evolutionary Algorithms. *10th International Conference on Control, Decision and Information Technologies (CoDIT)* (2024).
8. Kopanos GM, Méndez CA, Puigjaner L. MIP-based decomposition strategies for large-scale scheduling problems in multiproduct multistage batch plants: A benchmark scheduling problem of the pharmaceutical industry. *European Journal of Operational Research*, pp 644–655 (2010).
9. Beyer HG, Schwefel HP. Evolution Strategies A comprehensive introduction. *Natural Computing*, Vol. 1, pp 3–52, (2002).
10. Eiben AE, Smith JE. *Introduction to Evolutionary Computing*. Ed: 2, *Natural Computing Series*, Springer Berlin, Heidelberg (2015).

© 2025 by the authors. Licensed to PSEcommunity.org and PSE Press. This is an open access article under the creative commons CC-BY-SA licensing terms. Credit must be given to creator and adaptations must be shared under the same terms. See <https://creativecommons.org/licenses/by-sa/4.0/>

