Optimization and Engineering (2023) 24:447–468
https://doi.org/10.1007/s11081-021-09691-3

RESEARCH ARTICLE

Multi-objective optimization of energy-efficient production schedules using genetic algorithms

Tobias Küster[1] · Philipp Rayling[2] · Robin Wiersig[1] · Francisco Denis Pozo Pardo[1]

Received: 14 August 2020 / Revised: 16 September 2021 / Accepted: 16 September 2021 /
Published online: 4 October 2021

Abstract

The optimization of production schedules to be more energy efficient while still meeting production goals is a difficult task: How to schedule and distribute production tasks to meet production goals, while making best use of fluctuating energy market prices and availability of locally installed energy sources? Although a large body of related work exists in this domain, most of those seem to focus on individual aspects and not the whole picture. In this paper, a genetic algorithm for optimization of production schedules with respect to energy consumption, peak shaving, and makespan is presented, that also takes into account that tasks can be performed in different ways, having different characteristics. The algorithm has been successfully employed within the SPEAR project by applying it for optimization of an automotive production line and for the pathway of an automated guided vehicle.

Keywords Industrial optimization · Genetic algorithms · Energy modeling · Schedule optimization

This paper is based on research conducted in the SPEAR project, funded by the German Federal Ministry of Education and Research (BMBF), funding Reference Numbers 01IS17024G and 01IS17024H.

✉ Tobias Küster
tobias.kuester@dai-labor.de

Philipp Rayling
science@twt-gmbh.de

[1] DAI-Labor, Technische Universität Berlin, Ernst-Reuter-Platz 7, 10587 Berlin, Germany

[2] TWT GmbH Science & Innovation, Ernsthaldenstraße 17, 70565 Stuttgart, Germany

1 Introduction

In order to stay competitive in the modern globalized world, production processes have to be more agile, flexible, and faster-paced than ever before. And indeed, the rise of Industry 4.0 and increasing automation and digitization of production chains provide the opportunity to adapt and re-schedule processes on short notice. At the same time, today's variability in available energy sources and more flexible tariffs allow for shifting energy-intense tasks to more favorable times, and thus for the production to be both more economical and ecological. But production processes are also more complex, and often more fragile, with less slack, than ever before, making the timely optimization of such processes a challenging task.

The optimization of production schedules, as a variant of the prototypical *Job Shop Scheduling Problem*, has been the topic of numerous research papers (Fuchigami and Rangel 2018). The optimization goals usually are the minimization of the makespan or tardiness, but in recent times research also started to consider aspects such as energy consumption, availability of locally produced energy, and variable energy prices.

In the typical Job Shop Scheduling Problem, there are n jobs, each consisting of a number of o_n operations, each of which has to be executed on a specific machine, with the goal of minimizing the total time required for the operations such that no machine is used by two operations at the same time. Of course, there are also many different variations of the problem, such as the flexible job shop scheduling problem, or the flow shop scheduling problem (Dahal et al. 2007).

In this work, which is partially based on results previously introduced by Wiersig (2019), we will tackle a variant of the problem as presented in a recently completed research project, SPEAR.[1]

- There can be multiple *energy sources*, each with a variable price and availability, which can be used for representing power from locally installed sources or the grid, including tariffs with an upper limit on power.
- Products consist of *tasks*, which can be executed in different *task modes*, each of which can require a different machine, or have a different power consumption or duration.
- Order of execution and the concurrency of tasks can be constrained.
- Machines are grouped into *cells*, with the additional constraint that all the tasks required for creating one instance of a product have to be executed on machines from the same cell.
- Limited resources besides electric energy should be considered, e.g. the state of charge of buffer batteries, pressurized air, or raw materials.
- The goal of the optimization can be the reduction of the total makespan, the total energy usage and costs, the peak power consumption, or any weighted combination of those.

[1] SPEAR Project Website: https://spear-project.eu/.

The entire problem domain model resulting from those requirements will be explained in more details in Sect. 2. To the best of the authors' knowledge, those framing conditions, and in particular the concept of different task modes, have not been considered in related work until now (see Sect. 6).

Due to the large amount of variables and their complex inter-dependencies, the search space is very large and difficult to navigable efficiently using deterministic search and optimization approaches. Indeed, a study of several production scheduling systems found most using non-exact methods, especially for multi-criterion optimization (Fuchigami and Rangel 2018). In particular genetic algorithms have found much use in the field of production scheduling: They do not have many mathematical requirements to the underlying problem, can handle a variety of objective functions and constraints, and are well suited for multi-objective optimizations (Gen and Lin 2014).

With the inclusion of variable energy availability and costs, not only the allocation of machines and task modes to tasks and their ordering, but also the exact start times of all tasks have to be taken into account, resulting in a high number of possible combinations, and the dependencies between tasks can cause small changes to have large effects, possibly making the entire plan invalid. Hence, in this paper, we are using a genetic algorithm: A non-deterministic search algorithm inspired by the process of natural selection (Sakawa 2012), using concepts such as populations, crossover and mutation, fitness and selection to converge to a near-optimal solution. The used genetic representations and mutation operations minimize the chance for creating invalid intermediate schedules, and the fitness function can be tuned towards different optimization goals, as will be shown in Sect. 3.

The genetic algorithm described in this paper has been implemented as part of a production scheduling system in the course of the aforementioned research project (see Sect. 4). It was evaluated on different real-world problems presented by the project's industrial partners, which will be elaborated in Sect. 5. The main results of the paper are concluded in Sect. 7.

2 The production domain model

In the following, we will describe the production domain model used in this approach. The model can roughly be subdivided into three parts: The *production system*, consisting of the tasks and products and the infrastructure they can be produced with, the available *energy sources*, and different *constraints* that can restrict, e.g., the order in which tasks can be executed. Finally, all those parts are aggregated to an *optimization request*, yielding an *optimization result*, i.e. the production schedule.

2.1 Production system

The production system model provides information on all the tasks, task modes and machinery that are relevant for the optimization problem. This information can

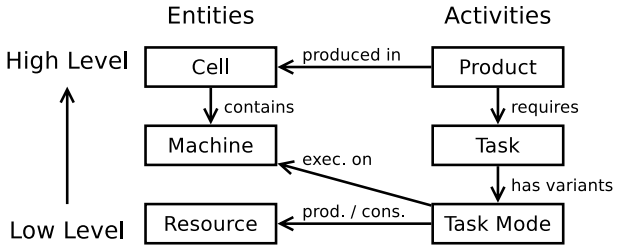


Fig. 1 Production system model, subdivided into entities and activities and showing their relations and dependencies

come from measurements performed on real hardware, or estimated by means of simulation of so-called *digital twins*.

The model can be subdivided into two sides: Entities, i.e. the installed machinery, and activities, i.e. the tasks that are to be performed. Also, those are subdivided into multiple levels: Production *cells* and *products* on a high level, and individual *machines* within a cell and *tasks* that make up those products on a low level. Each task has one or more *task modes* that describe different ways how the task can be executed and can require or produce different *resources* (Fig. 1).

One key distinguishing feature of this model is that each task can be implemented in different task modes, which could simply represent performing the same task on different machines, or different ways to manufacture the task that could, e.g., take more time while consuming less energy or vice versa, or consume the energy at a different point in time within the task.

Also, the model features resources that can be used to represent different intermediate products that are produced by one task and consumed by the next, or they can stand for environmental factors, such as one task (or task mode) producing waste heat that must not exceed a certain level, or the state-of-charge of a buffer battery. All resources must always remain within their respective minimum and maximum capacity during the entire production. While resources can be useful for modeling concepts and circumstances that could not be represented otherwise, in other settings they may not be needed and are thus optional. We will come back to resources later when discussing constraints.

2.2 Energy sources

The production system can be powered by different *energy sources*. Here, we do not make a strict distinction between different categories of energy sources, like energy bought from a provider or locally generated energy. Instead, each energy source has two attributes: A price and an availability, which can both vary for each point in time. Here, the price defines the amount of money to be paid for one unit of energy, and the availability denotes the units of energy available at that point in time. Note that units of time, energy, and price are chosen by the user and have to be used consistently throughout the entire optimization request.

This way, the same model can be used for socket power that has a (possibly variable) price and is (for all practical purposes) available in infinite quantities, as well as for limited locally installed "free" renewable energy. Further, negative availability can be used for energy sinks (parts of the production system that consume energy and can not be changed or shifted).

The model can also be used for capped energy tariffs, i.e. tariffs that offer a certain price but allow only a maximum of x Wh to be consumed, and will charge a higher price if that consumption is exceeded. Those can be modeled as two energy sources: The first with the lower price and an availability of x Wh, and another with the higher price and unlimited availability.

2.3 Constraints

The production system is subject to two types of *constraints*: General constraints and user-specified constraints. The first group includes, among others:

- No two tasks can be executed on the same machine at the same time;
- all tasks constituting one product must be performed in the same cell;
- the total energy consumption in any time step must not exceed the total available energy from energy sources;
- a task must be executed by a machine matching its task mode.

These constraints constitute the general "laws" of the system that always have to be respected.

Besides those, there are user-specified constraints, e.g., the production system may have a maximum peak energy usage (independently of what is available in individual energy sources) and a maximum total duration. Further, the tasks in a production system can have dependencies among each other, e.g., task A may have to be executed after task B, or at the same time as task C, or *not* at the same time as task D.

Finally, resource constraints can be used to specify more complex dependencies between tasks by having one task (or task mode) produce a resource that is then consumed by another task (or task mode). Among others, resources and resource constraints can be used for representing:

- resources like gas or pressurized air that are used for certain task modes,
- intermediate products passed from one stage of production to the next,
- waste products that may not be produced beyond a certain quantity,
- environmental states, e.g., the position or elevation of a table,
- the state-of-charge of, e.g., an electric forklift, or
- charging and discharging buffer batteries used for softening load peaks.

The aforementioned task-dependencies could also be represented using only resource-constraints, but dedicated task-dependencies make those both easier to describe in the model and more efficient to enforce or check.

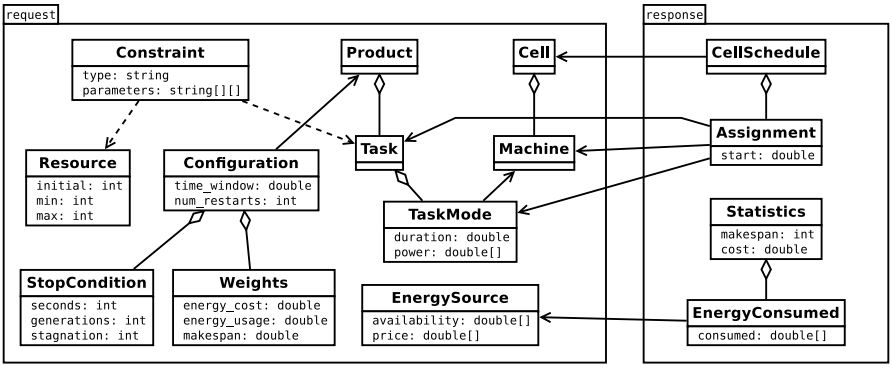


Fig. 2 Optimization model, divided into request and response part, slightly simplified. For readability, the actual Request and Response classes aggregating the different parts are not shown. All parts except for Configuration and Statistics can appear in multiple instances

2.4 Optimization requests and result

The actual *optimization request* includes, beside the above described production system, the energy sources with up-to-date availability and prices, and user-specified constraints, also a "configuration" object holding the number of products to complete, which then dictates how often to execute the individual tasks, as well as individual weights to the three competing optimization goals (the total makespan, energy costs, and peak energy consumption), and a stopping condition.

The *result* is a schedule for each production cell, describing for each of the tasks that make up the products (a) when the task is to be executed, (b) which task mode to use, and (c) on which machine to execute the task, as well as some statistics, like the total duration, energy consumption in each step, and total energy costs (see Fig. 2).

The request and result as well as the several features comprised in them have been modeled in an object-oriented way, allowing a straightforward implementation in Java and a transformation to readable JSON, as shown in Sect. 4. This request can then be passed to the optimization system, which will be described in more detail in the following section.

3 Optimization of production schedules

The optimization approach employed in this paper is based on Genetic Algorithms (GA), i.e. optimization algorithms that are inspired by genetics and the processes of evolution, mutation, and natural selection (Eiben and Smith 2003; Sakawa 2012). The basic working principle of a genetic algorithm consists of the following steps:

1. Creating an initial population of candidate solutions,

Table 1 Genetic representation and corresponding real-world concepts

Genetic concept	Meaning
Population	Current pool of candidate solutions
Phenotype	Production schedule (represented in the domain model)
Genotype	Production schedule (numeric repr. of variable parts)
Chromosome	Section representing a specific feature, e.g. pause times
Gene	Number representing, e.g., one specific pause time
Allele	E.g., pause time for one task in one spec. schedule

2. Randomly selecting, combining, and mutating individuals from that population,
3. Using a quality- or fitness-function to select the best individuals to carry over to the next generation.

Those steps are repeated, continuously improving the average quality of the population, until a predefined stopping-condition is met, e.g., a certain number of generations. Local minima can be circumvented by optimizing multiple populations and selecting the best result.

What sets genetic algorithms apart from other kinds of evolutionary algorithms is that the above steps are not carried out on the individuals directly. Instead, they are translated to a *genetic representation*, typically a sequence of numbers constituting the "genome" of the individuals, which is then mutated on a very low level, e.g., by changing or swapping individual numbers.

In the following, we will provide a detailed description of the genetic representation used for the production schedules, how their quality or fitness is determined, and how they are allowed to be mutated, or altered.

3.1 Genetic representation

Each candidate solution is represented as a genetic code, i.e. an array of integer numbers, subdivided into different "chromosomes" for different aspects of the schedule, which can then be mutated and recombined with a number of primitive and mostly domain-agnostic functions. Table 1 shows an overview of the terms used in the GA and the concepts they represent.

Given the information that makes up the schedule (not considering the derived information like total duration or total energy cost), a first naive representation would be to encode each task with three numbers on three different chromosomes: The starting time, the selected task mode, and the machine to execute the task.

The problem with this representation is that it is very easy to create invalid schedules. For instance, it is not easily possible to swap the order of two tasks to be executed on the same machine. First, one task would have to be moved "out of the way" before the other task can take its place and finally the first task can take the place of the second, otherwise there would be an immediate constraint violation. But even with this three-way-swap, each of the intermediate steps will

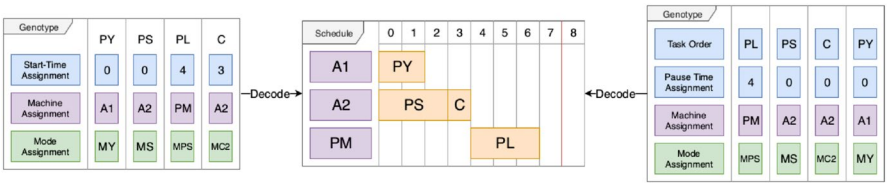


Fig. 3 Comparison of the initial start-time encoding (left) and the final order/pause encoding (right) representing the same schedule (centre)

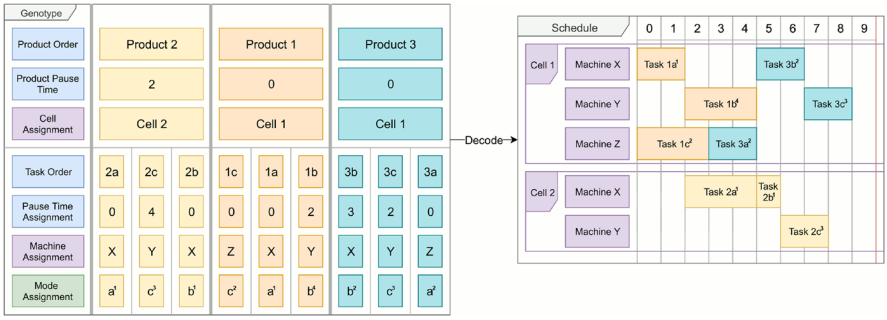


Fig. 4 Example genotype and corresponding schedule for multiple products and cells and the tasks and machines contained therein (based on Wiersig (2019))

likely have a lower quality than the original setup. Thus, the candidate schedules containing the partial swap will have a very high chance of being discarded before the swap is complete. Implementing this swap as an atomic operation is not trivial either, as the tasks might have different length and there might be other tasks in between.

An alternative representation, which was adopted in this work, is to replace the starting-time chromosome with two chromosomes for task ordering and pause times, where the former is a valid permutation of the tasks and the latter encodes the time for the task to *wait* after the last task occupying the same machine has been completed (or after the start of the corresponding product, if there is no prior task on the same machine). While this representation is slightly more complex and more computationally intensive to encode and decode, the benefit is that there can, by definition, be no two tasks occupying the same machine at the same time, and hence fewer candidate schedules will violate constraints. A swap operation as described above can be completed in a single mutation, even if the swap affects more than two tasks. Figure 3 shows a simple example comparing the two representations.

Similarly, on a higher level, the assignment of products to cells is encoded in another chromosome, as are the product ordering and product pause times, i.e. the times to wait before starting the next product (Fig. 4). Consequently, the tasks' start/pause times are always relative to the start/pause times of their product, and the machine used for a task is always taken from the machine pool of the corresponding product's cell.

3.2 Fitness function

The fitness function is subdivided into two parts: The hard score and the soft score. The hard score includes all the constraints that were defined in Sect. 2 (those that are not prevented by construction), while the soft score includes the makespan, total costs and energy usage, and the maximum power peak. The different aspects of both the hard and the soft score are combined as two weighted products, and then both are combined to the total score. The weighted products are calculated as (1), where $rating(s, c)$ is the rating of schedule s w.r.t. criterion c, and w_c is the weight given to that criterion.

$$score(s) = \prod_{c \in C} rating(s, c)^{w_c} \tag{1}$$

For both hard and soft scores, a lower score is considered better, i.e. if a schedule has a hard score of zero it has no constraint violations. A schedule with a lower hard score will always be preferred to a schedule with a higher hard score, independently of their soft scores. All individual ratings are ≥ 0, and, by adding a small $\varepsilon > 0$ to each, effectively > 0 so the product does not degrade to zero. The weights are normalized to have a sum of 1, such that, after subtracting the same ε from the final product, the score is 0 for a schedule without any defects.

3.3 Alteration and selection

In each generation g, the individuals of the population P_g are derived from the previous population P_{g-1} by selecting a random sample of n individuals to be kept without change, i.e. survivors, and a random sample of m individuals to be kept with random alterations, i.e. offspring. (2)

$$P_g = select(P_{g-1}, n) \cup alter(select(P_{g-1}, m)) \tag{2}$$

The following altering methods have been implemented:

- *Assign*: Assign a new value to a randomly chosen gene (e.g., pause times, task-mode, cell and machine assignment).
- *Swap*: Swap two values in a permutation chromosome (e.g., task ordering).
- *Shift*: Increase or decrease pause time for some task, consequently shifting the task itself and all tasks after it by some amount.
- *To-zero*: Set pause before a random task to zero; basically just giving a higher probability to one specific *assign* or *shift* alternation, but resulting in a significant performance boost in practice.

Recombination/crossover was tried in the form of partially matched crossover for permutations and single-point crossover for assignment chromosomes, but did not result in a significant improvement of quality.

4 Implementation

The optimization algorithm is based on the Jenetics library for Java.[2] Besides providing well-tested implementations for the basic genetic algorithm, Jenetics' extensive use of the Java Stream API and builder- and factory-patterns allows for a well-configurable and customizable optimization and parallel execution. Accordingly, the different mutation operations have been implemented as `Alterers` for Jenetics and the fitness function as a `Comparator` factory.

Besides the domain-agnostic alterers typical to genetic algorithms, the implementation also includes a variant of the *swap* alterer on task order permutations, using a graph-based algorithm to make sure that all task-dependency-constraints are still satisfied after the swap. This way, the number of mutations that are immediately discarded due to constraint violations is greatly reduced.

The domain model has been implemented using the Lombok[3] library, allowing the resulting classes to be very concise and consistent. Most of the relevant concepts have been explained in Sect. 2 and shall not be repeated here. The time series (for energy sources' price and availability, task modes' power consumption, and the overall schedule) can be represented either using a simple array of values, one for each discrete time step, or using a sparse navigable map, holding values from a certain starting time onward, using a `TreeMap`.

For running the optimization locally, and particularly for testing and debugging, a simple graphical user interface has been created using JavaFX.[4] Besides controls for loading a problem definition, configuring the different weights and for starting and stopping the optimization, the UI mainly consists of a large canvas for visualizing the currently best schedule, including the start times of the different tasks and the used task modes, the machines they are executed on, and the total energy consumption over all time steps.

The optimization has been wrapped into a Spring Boot[5] application, providing REST services for configuring and running the optimizations, and made available as a Docker image at https://hub.docker.com/r/dailab/spear-rest-interface. Parameters and results are encoded as JSON strings, making the services easily usable from other applications or using a simple Swagger[6] Web UI. A publicly accessible instance of this application is running at https://spear.aot.tu-berlin.de:8082/. The code has been made available as open source (ITEA 2021) and can now be found in the GitLab repository at https://gitlab.dai-labor.de/spear/spear-optimisation.

Besides the actual optimization and its implementation, another notable goal of the SPEAR project was the specification of a generic and reusable interface for optimization services. Based on the above REST interfaces and JSON models, a further consolidated version has been worked out together with the project partners, for

[2] Jenetics (v 4.3.0): https://jenetics.io/.
[3] Lombok (v 1.18.6): https://projectlombok.org/.
[4] JavaFX (v. 11.0.2): https://openjfx:io/
[5] Spring Boot (v 2.0.5): https://spring.io/projects/spring-boot.
[6] Swagger (v 2.9.2): https://swagger.io/.

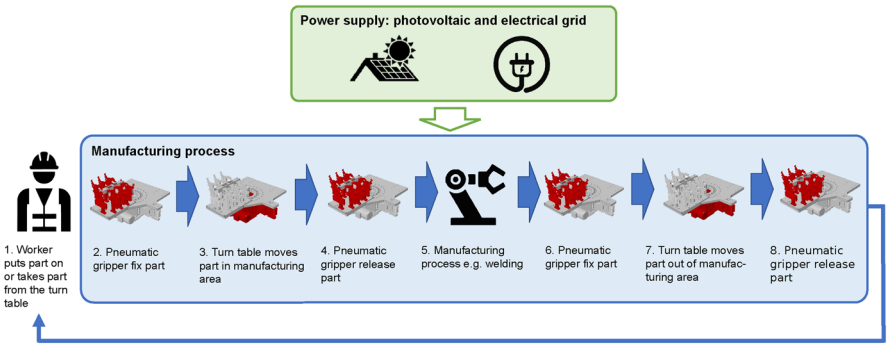


Fig. 5 Schematic illustration of the manufacturing process

controlling and optimizing processes in the textile and food industries. This version of the API as well as an extensive description can be found at http://spear.aot.tu-berlin.de:8080/.

The optimization algorithm and API were used as part of a larger production scheduling system, together with a data provisioning backend and web-based user dashboard provided by the project partners.

5 Evaluation

In this section, we will evaluate the optimization using two examples based on real-world processes: A production process involving a turn table with gripper, which is part of an automotive manufacturing line, and a process including an automated guided vehicle (AGV). Both examples are part of the research project SPEAR and exploit the advantages of having appropriate digital twins capable of describing the different tasks. In this manner, a large number of task variants can be considered with little effort. This approach is especially advantageous during virtual commissioning, where measurements on real hardware are not yet possible. Also, for both examples measured energy values were available for validation. The processes will be optimized for makespan, costs, and energy usage with different relative weights.

5.1 Turn table with gripper

For security reasons, automated production cells are commonly enclosed by a safety fence preventing human-machine collisions. The use case analyzed here makes use of a turn table that can safely bring a part in- and outside the enclosed cell by rotating $\pm 180°$. The turn table is driven by a bevel gear (SEW KTF57/R) and an asynchronous motor (SEW DRL100L4BE45) with six pneumatic grippers (Festo ADVU-12-50). The electrical power for the machine is provided by a photo voltaic plant (PVP) with a limited availability but assumed as free of charge (installation, amortization and running costs are neglected) and from an electrical grid with

unlimited availability but energy costs. The process consists of several tasks, which are exemplary shown in Fig. 5 and are described in the following:

1. Outside of the production cell, a worker puts the unmachined part on the turn table
2. The part is fixed by the pneumatic gripper
3. The turn table rotates by 180°, bringing the unmachined part inside the production cell
4. The gripper releases the part
5. The part is taken by robots for the further manufacturing process inside the cell like deep drawing and attachment of additional components by welding; upon achieving these tasks, the robot puts the part back on the turn table
6. The gripper fixes the part again
7. The turn table rotates back by 180° bringing the manufactured part outside the production cell
8. The gripper releases the part again and the worker takes the finished part from the turn table

The components were modeled in the object-oriented and equation-based language Modelica[7] (Mattsson et al. 1998; Fritzson and Engelson 1998; Fritzson 2010) making use of the Modelica Standard Library. The models were validated against measured energy curves taken from the cell provided by FFT Produktionssysteme GmbH & Co. KG[8] as part of the SPEAR project. To extend the modularity, these digital twins have been exported as functional mock-up units (FMUs) (Blochwitz et al. 2011, 2012; Modelica 2019). A large set of consumption profiles (task modes) have been computed by varying relevant parameters in the digital twins using the PyFMI[9] simulation environment. In this use case, the task modes for each of the components differ on their execution velocity, as will be explained later. The resulting consumption profiles are expressed as time series of power.

The resulting power curves for the non-optimized process are plotted in Fig. 6a. In red the consumed power for the manufacturing stages are depicted. For the actual manufacturing process inside the cell a generic constant power consumption of 10 kW is assumed as no real values can be given due to intellectual property protection considerations. In green the power from the photo voltaic plant is plotted. If the power consumption is higher than the energy from the PVP (negative sign), additional power is taken from the electrical grid, whereas unused photo voltaic power is fed back in the power system (positive sign). This power, which is a balance of the green and red curve, is plotted in blue. The resulting energy curve is plotted as blue dashed line. The whole non-optimized manufacturing process takes approximately 46.5 s per part and a total energy of 5.9 kJ is taken from the electrical grid.

[7] Modelica: https://www.modelica.org/modelicalanguage.
[8] FFT Produktionssysteme GmbH & Co. KG: https://www.fft.de.
[9] PyFMI: https://www.jmodelica.org/pyfmi.

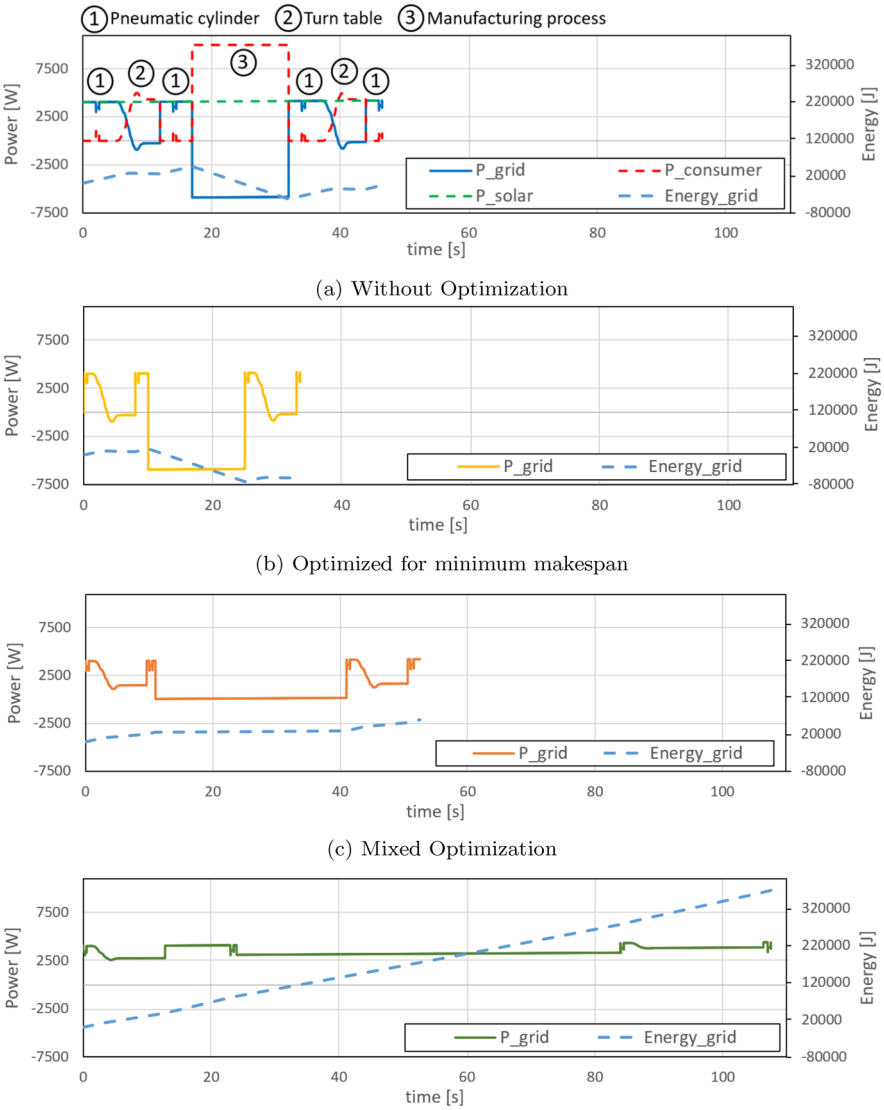
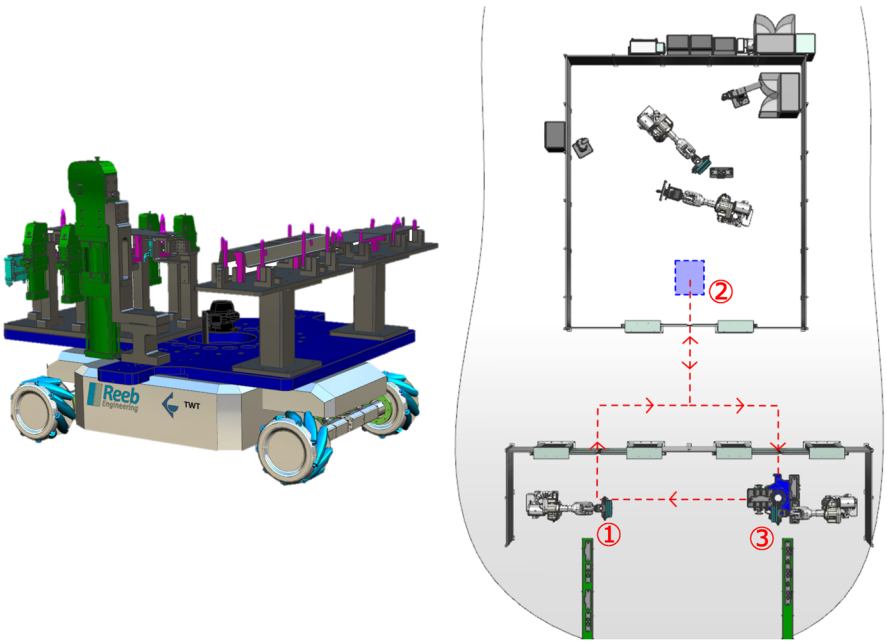


Fig. 6 Comparison of **a** original process, **b** optimized for makespan, **c** optimized for makespan and costs, and **d** optimized for minimum power consumption from the electrical grid for the turn table with gripper

For the optimization process different production modes were considered, corresponding to different production speeds, i.e. 25%, 50%, 75% and 100% of the maximum production speed. For validation, the process was optimized for makespan, costs and energy usage with different relative optimization weights. In Fig. 6 the original process without optimization (Fig. 6a), the process purely optimized for makespan (Fig. 6b), a mixed optimization for makespan and energy consumption of



(a) Rendering of the AGV (b) Pathway of the AGV in the plant

Fig. 7 Depiction of the AGV, the plant and the pathway

energy from the electrical grid with equal weights (Fig. 6c), and purely optimized for minimizing the power consumption from the electrical grid (Fig. 6d) are depicted beside each other for comparison. Additionally, the curves for the energy taken from or fed into the electrical grid are depicted as blue dashed lines. The manufacturing process optimized for makespan takes only 33.5 s, but the power consumption from the electrical grid is increased to 60.0 kJ. When optimizing for minimum costs and energy usage, the process takes 107.5 s and a total energy of 369.3 kJ is fed into the energy grid due to the PVP. Mixed optimizations lies in between these extreme cases. As a compromise the process takes 51.5 s and a total energy of 55.7 kJ is fed into the energy grid. This means for less than 10% longer makespan 61.6 kJ can be saved, which is approximately 30% of the overall power consumption. The results show that the presented algorithm is capable of optimizing the process in the machine for different optimization goals.

5.2 Automated guided vehicle

As a second use case, the driving path for an automated guided vehicle (AGV) is optimized. The AGV, which was designed and developed by Reeb-Engineering

GmbH[10], is depicted in Fig. 7a. It serves as an alternative for the turn table, described in Sect. 5.1, supplying the manufacturing cell with new parts and transporting the processed parts away automatically. For this example, the velocity and drive modes for the path shown as red dotted line in Fig. 7b were optimized.

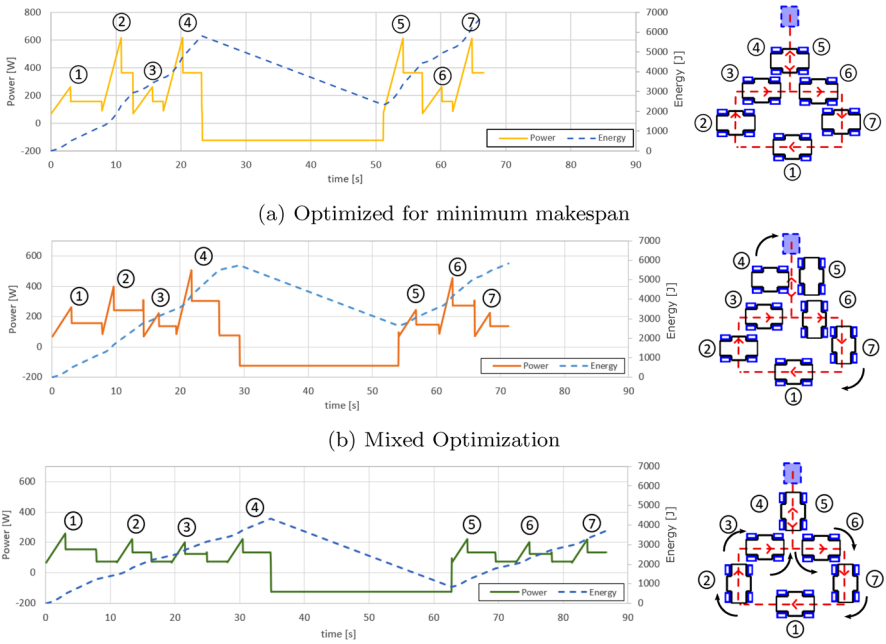
The AGV considered here moves on four Mecanum wheels that allow holonomic, omnidirectional movement in the plane. Thus, an infinite number of displacement combinations (directions, orientations and velocities of the movement) exist for a given target position. To limit the number of possible combinations, only longitudinal and transverse movement as well as rotation is considered. In position 1 an unmanufactured part is loaded onto the AGV. Afterwards the AGV drives to position 2, where the part is taken from it automatically and further processed. During the manufacturing, the AGV battery is charged by induction marked by the blue rectangle in Fig. 7b. Then the AGV drives to position 3 where the processed part is taken from the AGV for further processing. In total the pathway consists of six different positions (two of which are visited twice, hence seven segments) with two driving modes (longitudinal and transverse) and ten different maximum velocities for each driving mode as well as five charging speeds. The rotation is automatically included where it is needed to achieve the target orientation. This leads to $(2 \cdot 10)^7 \cdot 5 = 6.4 \cdot 10^9$ different possibilities in total.

Similar to the first use case, the modeling of the AGV has been performed in the Modelica Language. The system is divided into four main components: a) a target coordinate system, b) a movement controller, c) the motion prediction describing the transients of the different movements, and d) a simplified battery model. The vehicle motion has been modeled based on velocity, position and energy consumption measurements on the real device provided by Reeb-Engineering GmbH as part of the SPEAR project. From these measurement, it can be seen, for example, that longitudinal movements are energetically more efficient than transverse movements. For visualization and validation the possibility to simulate FMUs in Siemens NX MCD™[11] was used and a video of the resulting process is provided by Reeb-Engineering GmbH (2020).

The results of the optimizer are plotted in Fig. 8. The solid lines are the power curves and the dashed lines are the corresponding energies. While the part is processed, which takes 28 s, the AGV battery is charged, which can be seen by the negative sign of the power curves as well as the negative slope of the energy curves. In Fig. 8a the path optimized for minimum makespan is shown, which takes about 66 s to complete a round. To get a minimum makespan the segments 2, 4, 5 and 7 are driven transversely, because no rotation is needed. Also, as expected, the optimizer proposes to use the maximum velocity for each segment. This combination of modes consumes an energy of 10.4 kJ in total. In Fig. 8b the result of a mixed optimization is plotted, where the energy consumption and minimum makespan are equally weighted. This process takes approximately 71 s and an energy of 8.9 kJ is

[10] Reeb-Engineering GmbH: https://www.reeb-engineering.de.

[11] Siemens NX: https://www.plm.automation.siemens.com/global/en/products/mechanical-design/mechatronic-concept-design.html.



(a) Optimized for minimum makespan

(b) Mixed Optimization

(c) Optimized for minimum costs and energy usage

Fig. 8 Comparison of the process **a** optimized for makespan, **b** optimized for makespan and energy usage, and **c** optimized for energy usage

used. To reduce the power consumption, between segments 4 and 5 as well as 7 and 1 rotations occur, and the path is driven with lower velocities than the path presented in Fig. 8a. Because of the rotation between segments 4 and 5, the available charging time is reduced (rotation time plus charging time equals 28 s). Finally, in Fig. 8c the path for minimum costs and energy usage is plotted. For all way points, except the one between 4 and 5, the AGV rotates, allowing to only drive longitudinally. This leads to a longer makespan of 86.7 s in total and energy consumption of 7.2 kJ. Contrary to initial expectations, the optimum driving mode for minimum energy usage does not employ minimum driving velocities due to the fact that lower velocities use temporarily lower energy, but for longer time, leading to a higher energy consumption in total. This observation underlines the necessity of sophisticated optimization approaches, like the one presented in this work. The results also demonstrate that, comparing the paths for minimum makespan and the mixed optimization, only 6% longer makespan can save approximately 14% of the energy consumption.

For validation, also all possible combinations were evaluated to make sure to get the global minimum of makespan and power energy usage and not just a local minimum. This kind of "brute-force" optimization took seven days on a single-core 2.4 GHz CPU, compared to approximately one hour with the presented framework based on genetic algorithms. It was shown that both approaches (brute-force and genetic algorithms) give the same results and hence that the presented results are global minima and not just local minima, but the presented approach took just about

1% of the time of the brute-force approach. Thus, this example can be solved efficiently by genetic algorithms.

Other approaches use Jacobian matrices to avoid evaluating all permutations of possibilities (Braun et al. 2011; Åkesson et al. 2012). However, not all permutations are valid and the numerical effort would be tremendous. On the other side, sequential optimization of intermediate segments or single movements might not lead to a global optimum.

6 Related work

As mentioned in the introduction, there is a very large body of related work for genetic algorithms and other optimization techniques applied to the job-shop scheduling problem, dating back many years—see, e.g., Wall (1996) for an early overview, or Gen and Lin (2014) and Fuchigami and Rangel (2018) for a more recent perspective. However, the authors are not aware of any existing work adhering to the same constraints and framing conditions as in this work, in particular the possibility of different task modes, resources representing constraints between those task modes, and variable energy sources.

In the following, we will discuss a number of related works—including one article of previous work of the authors—that are close to the problem, covering significant subsets of the above aspects or using different constraints. Table 2 gives a comparison of the different papers and their properties.

- Bukata et al. (2017) optimize the energy consumption of robotic cells by using different speeds, positions, or power-saving modes for the individual robots in the cell. However, they do not consider variable prices or limited availability of energy.
- Shrouf et al. (2014), on the other hand, take the time-of-use energy price into account for reducing the costs by means of a genetic algorithm. However, while they also consider different power modes, they only consider a single-machine use case and also do not regard limited availability of energy.
- Zhou et al. (2018) use an evolutionary algorithm to bi-objectively optimize batch processing machine schedules for makespan and total energy cost, also considering different energy prices.
- Waschneck et al. (2018) applied Google DeepMind's Deep-Q-Network agent algorithm for Reinforcement Learning to schedule a production system. The agents are trained to optimize different parts of the schedule with different optimization criteria and constraints. They did not schedule for any energy-related goals, but since their solution is quite general in regard to the optimization goals it could probably be used to optimize for energy consumption, availability and price.
- Lee et al. (2017) propose a dynamic control algorithm that multi-objectively optimizes a single-machine use case in regard to energy price while penalizing for earliness or tardiness of certain jobs. They evaluated their approach by the usage of real energy and machining parameters of a Haas milling machine,

Table 2 Comparison of related work, based on Wiersig (2019)

	Multi-object.	Global optim.	Energy cons.	Energy cost	Energy avail.	Evolut. algor.
Bukata et al. (2017)		✓	✓			
Shrouf et al. (2014)			✓	✓		✓
Zhou et al. (2018)	✓		✓	✓		✓
Waschneck et al. (2018)		✓	(✓)	(✓)	(✓)	
Lee et al. (2017)	✓		✓	✓		
Küster et al. (2013)	✓		✓	✓	✓	✓
Algorithm presented in this work	✓		✓	✓	✓	✓

which resulted in an overall lower total cost on average compared to the metaheuristic approach.

- Küster et al. (2013) used an evolutionary algorithm to optimize manufacturing processes according to various criteria, including energy consumption, cost and makespan. The production system model is divided into activities (production tasks and secondary processes) and resources (raw materials, intermediate products, and others). They also consider both variable energy prices and limited availability of locally produced energy. However, while the model is very flexible and domain-agnostic, the lack of a concept for different modes for the same task makes it impossible (or very cumbersome, using a combination of 'primary' and 'secondary' activities) to represent certain problems.

Similar approaches can also be found, among others, in Bernik and Bernik (2007), who use genetic algorithms for optimizing production schedules, but do not account for energy aspects, or Schreiber et al. (2009), optimizing for production targets and lot-sizes.

As can be seen, some of the state of the art is close to the approach presented in this work, while using a variety of techniques from mathematical optimization via stochastic or evolutionary algorithms to machine learning. Still, while some of the existing approaches could probably be extended in this direction, the exact conditions encountered in the SPEAR project were not yet tackled in related work. The major weak point of our approach—as with other stochastic approaches—is that there is no guarantee that the global optimum is found, although the chances can be improved with a variable mutation rate and random restarts.

7 Conclusion

In this work, a versatile framework for optimizing production schedules is presented. The underlying model includes concepts such as products, tasks and task-modes, machines and cells, resources for modeling dependencies between tasks, and variable energy sources, allowing a wide range of production scenarios to be modeled and optimized. The optimization, based on a genetic algorithm, allows to optimize for makespan, energy consumption or cost, or a mixture/compromise of those goals.

The model, optimization, and API have been developed in the course of the SPEAR project, involving several partners from research and different sectors of industry, thus ensuring the applicability and versatility of the approach. The implementation of the optimization algorithms and API (see Sect. 4) have been provided as both open source code and as Docker images ready for deployment.

To demonstrate the broad range of applications, the optimization of a machine from an automotive manufacturing line and the optimization of the pathway of an AGV are shown. In both cases, the optimization was able to find a non-trivial solution trading a slight increase in makespan for a significant gain in energy efficiency. From the results we draw the following four main conclusions:

- The presented framework is capable of efficiently optimizing a wide range of production tasks.
- Its flexibility enables the user to optimize complex tasks with several energy sources and constraints. Due to the usage of resources, even dynamic processes like charging and discharging of batteries can be included in the optimization process.
- The framework is well-suited for typical engineering use-cases, as demonstrated by two application examples.
- While the employed genetic algorithm can, by its nature, not guarantee to reach the optimal solution, it yielded reliable near-optimal results in practice.

This work showed the capability of the presented approach to create production schedules that are competitive w.r.t. production time while being both, more economical, and, by preferring locally installed green energy sources, more ecological.

Acknowledgements This paper is based on research conducted as part of the SPEAR project, funded by the German Federal Ministry of Education and Research (BMBF), funding Reference Numbers 01IS17024G and 01IS17024H. The authors of this paper want to express their gratitude towards the entire SPEAR project consortium, in particular to Alejandro Cardenas from TWT GmbH Science & Innovation for valuable feedback to the paper, the management of TWT GmbH Science & Innovation for supporting this research, the partners of FFT Produktionssysteme GmbH & Co. KG and Reeb-Engineering GmbH, especially Thomas Bleistein, for their contributions to the evaluation. We thank Anton Strahilov from EKS InTec GmbH for his efforts as the project lead, as well as the ITEA Office and the German Federal Ministry of Education and Research (BMBF) for making the research conducted within the SPEAR project possible.

Author Contributions The authors have contributed in the following ways to the research and the manuscript: FDPP created the initial concept for the optimization model. RW refined the concept and created the implementation in the course of his master's thesis. PR conducted the evaluation of the algorithm on different scenarios. TK refined and extended the optimization model and implementation. All listed authors have contributed to and reviewed the final manuscript. Further, Alejandro Cardenas of TWT GmbH Science & Innovation provided valuable feedback and helped with formulations in various parts of the manuscript. Also, the partners of FFT Produktionssysteme GmbH & Co. KG and Reeb-Engineering GmbH, especially Thomas Bleistein, contributed data on their processes and graphics for the evaluation.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Åkesson J, Braun W, Lindholm P, Bachmann B (2012) Generation of sparse Jacobians for the Function Mock-up Interface 2.0. In: Proceedings of the 9th international Modelica conference, pp 185–196

Bernik I, Bernik M (2007) Multi-criteria scheduling optimization with genetic algorithms. In: Proceedings of the 8th WSEAS international conference on evolutionary computing, world scientific and engineering academy and society (WSEAS), Stevens Point, Wisconsin, USA, pp 253–258

Blochwitz T, Otter M, Arnold M, Bausch C, Clauss C, Elmqvist H, Junghanns A, Mauss J, Monteiro M, Neidhold T, et al. (2011) The functional mockup interface for tool independent exchange of simulation models. In: Proceedings of the 8th international modelica conference, Linköping University Press, pp 105–114

Blochwitz T, Otter M, Akesson J, Arnold M, Clauss C, Elmqvist H, Friedrich M, Junghanns A, Mauss J, Neumerkel D, et al. (2012) Functional mockup interface 2.0: the standard for tool independent exchange of simulation models. In: Proceedings of the 9th international Modelica conference

Braun W, Ochel L, Bachmann B (2011) Symbolically derived Jacobians using automatic differentiation-enhancement of the OpenModelica compiler. In: Proceedings of the 8th international Modelica conference; March 20th-22nd; Technical Univeristy; Dresden; Germany, Citeseer, 063, pp 495–501

Bukata L, Sucha P, Hanzalek Z, Burget P (2017) Energy optimization of robotic cells. IEEE Trans Ind Inf 13(1):92–102. https://doi.org/10.1109/TII.2016.2626472

Dahal K, Tan KC, Cowling PI (2007) Evolutionary scheduling. Springer

Eiben A, Smith J (2003) Introduction to evolutionary computing. Springer

Fritzson P (2010) Principles of object-oriented modeling and simulation with Modelica 2.1. Wiley

Fritzson P, Engelson V (1998) Modelica-A unified object-oriented language for system modeling and simulation. In: European conference on object-oriented programming, Springer, pp 67–90

Fuchigami HY, Rangel S (2018) A survey of case studies in production scheduling: analysis and perspectives. J Comput Sci 25:425–436. https://doi.org/10.1016/j.jocs.2017.06.004

Gen M, Lin L (2014) Multiobjective evolutionary algorithm for manufacturing scheduling problems: state-of-the-art survey. J Intell Manuf 25(5):849–866. https://doi.org/10.1007/s10845-013-0804-4

ITEA (2021) SPEAR optimisation algorithm for production processes now available as open source. Tech. rep., ITEA, https://itea4.org/news/spear-optimisation-algorithm-for-production-processes-now- available-as-open-source.html, Accessed on 03 sept, 2021

Küster T, Lützenberger M, Freund D, Albayrak S (2013) Distributed evolutionary optimisation for electricity price responsive manufacturing using multi-agent system technology. Int J Adv Intell Syst 7

Lee S, Do Chung B, Jeon HW, Chang J (2017) A dynamic control approach for energy-efficient production scheduling on a single machine under time-varying electricity pricing. J Clean Prod 165:552–563. https://doi.org/10.1016/j.jclepro.2017.07.102

Mattsson SE, Elmqvist H, Otter M (1998) Physical system modeling with Modelica. Control Eng Pract 6(4):501–510

Modelica (2019) FMI specification 2.0.1. Tech. rep., Modelica Institution, https://www.fmi-standard.org, Accessed on 03 sept, 2021

Reeb-Engineering GmbH (2020) Energiesimulation fahrerloses Transportsystem FTS, AGV Simulation (video). https://www.youtube.com/watch?v=D-RInfyVhXo, Accessed on 03 sept, 2021

Sakawa M (2012) Genetic algorithms and fuzzy multiobjective optimization. Springer

Schreiber P, Vazan P, Tanuska P, Moravcik O (2009) Production optimization by using of genetic algorithms and simulation model. In: Katalinic B (ed) DAAM International Scientific Book 2009. DAAM International

Shrouf F, Ordieres-Meré J, García-Sánchez A, Ortega-Mier M (2014) Optimizing the production scheduling of a single machine to minimize total energy consumption costs. J Clean Prod 67:197–207. https://doi.org/10.1016/j.jclepro.2013.12.024

Wall MB (1996) A genetic algorithm for resource-constrained scheduling. Ph.D. thesis, Massachusetts Institute of Technology

Waschneck B, Reichstaller A, Belzner L, Altenmüller T, Bauernhansl T, Knapp A, Kyek A (2018) Optimization of global production scheduling with deep reinforcement learning. Proc CIRP 72:1264–1269. https://doi.org/10.1016/j.procir.2018.03.212

Wiersig R (2019) Applying genetic algorithms for multi-objective optimisation of energy-efficient production schedules. Master's thesis, Technische Universität Berlin

Zhou S, Li X, Du N, Pang Y, Chen H (2018) A multi-objective differential evolution algorithm for parallel batch processing machine scheduling considering electricity consumption cost. Comput Oper Res 96:55–68. https://doi.org/10.1016/j.cor.2018.04.009