MASTER

Solving a multi-objective optimization scheduling problem in a high mix low volume jobshop

Weijers, Eric T.

Award date:
2023

Department of Industrial Engineering and Innovation Sciences
Operations Planning Accounting and Control Research Group

MSc. Thesis

Solving a multi-objective optimization scheduling problem in a high mix low volume jobshop

By
E.T. (Eric) Weijers, 0893663

Supervisors:
dr. C. Fecarotti, TU/e
dr. A.E. Akçay, TU/e
dr. Q.V. Dang, TU/e
F. van Wylick, Nooteboom trailers BV
C. Uijlenbroek, Nooteboom trailers BV

Eindhoven, February 2, 2023

Contents

1 Introduction

Nowadays, a growing number of Small, Medium and Micro enterprises (SMMEs) is moving towards the implementation of smart manufacturing and industry 4.0 to improve their performance. Industry 4.0 entails transforming existing manufacturing systems to Smart Manufacturing Systems (SMS) through the support of automation, data exchanges, cyber-physical systems (CPS), cloud computing, robotics, big data, artificial intelligence, internet of things (IoT) and semi-autonomous industrial techniques [1]. In a research amongst 79 companies of different sizes, the authors in [2] observe significant improvements in the performances of companies implementing industry 4.0 components. In particular, increased production, increased productivity, decrease in costs, increase in capacity utilization rates, increase in production speed and increase in both product quality and the quality of workplace safety for employees were among the most important improvements. The authors in [3], based on their research on SMMEs, conclude that the the same improvements can be achieved for SMMEs.

While industry is moving towards smart manufacturing and industry 4.0, customer demand is shifting towards more customized products, thus causing product variety to increase. As a result, manufacturing companies are shifting their production strategy from make-to-stock (MTS) to make-to-order (MTO) production [4]. MTO production implies a high-mix, low-volume production environment. A manufacturing system capable of coping with such an environment is a jobshop. A jobshop is typically characterized by a highly diverse product mix. Each product follows its own manufacturing path through the jobshop, with unique setup times and cycle times per work center. A work center consists of one or more machines with identical process capabilities. Each machine has a finite capacity per day. Demand variability per product is high. When demand for a product occurs, a job is initiated. Each job has a unique due date and lot sizes per job can vary between 1 and over 100 products. Due to the high product demand variability, the diversity in routings and the finite machine production capacity, production bottlenecks can shift over time to different work centers [5].

The aforementioned jobshop characteristics results in complex production scheduling and control. Scheduling plays a crucial role in the competitiveness of a jobshop. A good schedule will maximize production efficiency and concurrently minimize various costs [6]. The Classical Job Shop Scheduling Problem (JSSP) is a well known problem in literature, next to which a number of variations exist, as shown in the review in [7]. The most important types are Dynamic JSSP, Flexible JSSP, Distributed JSSP, JSSP considering machine availability, JSSP considering setup times, JSSP with non-deterministic or non-constant processing time, JSSP with dual-resource constraints, JSSP considering energy and pro-environment and JSSP considering batches. Within the last variant, we can distinguish between the parallel batch problem and the serial batch problem or batch decision scheduling problem. Under parallel batching, multiple jobs can be processed simultaneously on the same parallel batch processing machine. The processing time of a batch is dependent on the individual jobs present in the batch and equals the maximum of individual processing times [8]. Examples of parallel batching are diffusion in semiconductor manufacturing or heat treatments in metal fabrication [9]. Under serial batching, jobs with common machine requirements are processed consecutively on a given machine. The processing time of a batch is dependent on the individual jobs present in the batch and equals the sum of individual processing times [9]. Examples of serial batching are the extrusion of cylindrical aluminum ingots in the aluminium industry [10] or the laser cutting of a sheet metal plate in a sheet metal jobshop [11]. In the remainder of this paper, we will focus on serial batching.

Compared to the classical JSSP, a JSSP considering batches adds additional complexity to the scheduling task due to the batching problem. Not only should the batching problem be solved, but it also introduces a second objective thus making the problem a multi-objective

optimisation (MOO) problem. In a JSSP, scheduling is based on one or more criteria, among which time, job number, cost, revenue, energy and pro-environment [7]. In serial batching problems, minimization of setup times [12], maximum earliness, total number of tardy jobs [10] and waste material [13] are common objectives. When the aim is to minimize waste material, the batching problem is formulated as a two-dimensional bin packing problem (2D-BPP), where a set of two-dimensional rectangular items need to be packed into the minimum number of two-dimensional rectangular bins without overlap. The 2D-BPP is a NP-hard problem [14].

In this research, the MOO problem within the sheet metal jobshop at Nooteboom Trailers BV is considered. Nooteboom Trailers BV is a manufacturer of trailers for abnormal road transportation producing around 750 trailers per year. Within the jobshop at Nooteboom, jobs having the same material and thickness are batched on a sheet metal plate to be cut by the laser cutter of TRUMPF SE + CO. KG. The main objective during batching is to maximize the number of jobs per batch to minimize waste material, thus a 2D-BPP problem needs to be solved. After cutting, jobs are manually picked out of a plate and transported to one of the preceding work centers. Depending on the operations needed, a job visits the required work centers. Next to the laser cutter work center which contains one laser cutter, and a picking work center which contains one picking area, the jobshop contains a hole tapping work center containing one tapping machine of IMM, a milling work center containing one milling machine of Anayak, a bending work center containing two press brakes of TRUMPF SE + CO. KG and finally a logistics work center. For all work centers, a production sequence of jobs needs to be determined, thus a JSSP needs to be solved. The main objective during scheduling is to maximize delivery performance by minimizing tardiness of jobs.

This paper introduces an optimisation model based on the metaheuristic Generalized Differential Evolution 3 (GDE3). GDE3 is selected for its proven performance regarding Pareto set generation. The use of a Pareto set enables a decision making process in which the trade-off between conflicting objectives can be decided upon. The main contribution of this paper is to present a modeling approach and solution generation method enabling a decision making process to select a optimal production schedule within a JSSP considering batches. From a modeling perspective, the proposed model adds value by incorporating a reformulation of the batching constraint increasing batch feasibility, setup times, non-homogeneous material thickness and both machine assignment and batch processing time calculation during the optimization process. From a solution generation perspective, an a posteriori method for solution generation is used. Benefits of this method are the fact that domain knowledge is not required prior or during the optimization process in combination with good performance regarding Pareto set generation. The method is guaranteed to find a uniformly distributed Pareto optimal solution set (providing a higher amount of solutions to the decision maker compared to an a priori method) and the non-convex regions of the Pareto optimal set can be obtained, resulting in higher solution quality compared to an a priori method. The final contribution of this paper is to use GDE3 for solving the MOO problem, which has not been used to solve a JSSP considering batches.

This paper is structured as follows. After reviewing relevant literature regarding the JSSP considering batches and metaheuristic algorithms with an emphasis on Pareto set generation in section 2, a formal problem description is presented in section 3. Then section 4 provides the mathematical formulation of the model. After discussing the solution approach in section 5, numerical results of a case study at Nooteboom Trailers BV are presented in section 6. The paper is concluded with managerial insights in section 7 and conclusions and recommendations for future research in section 8.

2 Literature review

In literature, both the batching problem and flexible JSSP have received much attention. However, the JSSP considering batches, which is a combination of both problems, is only studied marginally. Amongst the papers discussing this problem, different modeling approaches are proposed. While authors usually model the JSSP considering batches as a MOO problem, some authors simplify and transform the MOO problem at hand into a classical single objective optimization (SOO) flexible Jobshop Scheduling Problem (FJSP). After discussing literature on JSSP considering batches, this study will focus on different MOO modeling approaches and in particular on solution generation with an emphasis on Pareto set generation. Metaheuristics are capable of solving large MOO problems and finding Pareto sets of good quality in a reasonable time frame. Therefore, literature on metaheuristic algorithms is discussed.

In [15], the authors formulate the JSSP considering batches as a capacitated vehicle routing problem. Metal sheets with a limited area are represented as trucks with a limited capacity. Parts with a certain surface are represented as different customers with a specific demand, while set-up times between production layouts within the bending process are represented as travel distances between the different customers. The problem is then modelled using Linear Programming (LP). The objective function minimises the single objective of total sequence-dependent set-up time between the production layouts of the different parts. In order to sequence the metal sheets for cutting such that setups are minimized at the press brake, a variable neighbourhood search (VNS) is proposed. Further improvements to the model entail the inclusion of rush orders, different routings (eg. jobs which only need to be processed at the cutting process) and the capability of handling multi-machine instances.

In a more recent contribution [16] a combined processing constraint (batching constraint) in combination with a virtual operation (eg. when a batch is being processed, multiple jobs are processed at once. This is modeled as one virtual operation and not a separate operation for each job within the batch) to simplify and transform the MOO problem at hand into a SOO FJSP is proposed. Minimizing make span is considered as objective. A Multi-agent system (MAS) is used for building a schedule and solving the FJSP. In particular, a contextual bandit (CB), which is a form of reinforcement learning (RL), is used to model the scheduling process. CB only contains one state per episode and only affects immediate reward. The rewards depend on the context information provided at each time slot. Each job agent must learn to select the best dispatching rules according to the environment state. To do so, the decision process of a job agent consists of two stages, being machine selection and buffer job sequencing. Next to a job agent per job, the model contains one manager agent and a machine agent per machine.

In accordance with [16], the authors in [17] use a virtual operation to simplify and transform the MOO problem at hand into a SOO FJSP. To solve the FJSP, a Genetic Algorithm (GA) is proposed. GA, combined with a centralized cloud solution, enables separating the optimization problem and solution generation part. By doing so, different objectives can be pursued using the same solution generation part but with tailored fitness functions. In their paper, the authors consider minimizing make span as objective. However, minimizing waste material at the cutting process could be integrated in the GA as well by altering the fitness function. Depending on the production process at hand, the authors advice to use either minimizing make span or waste material as objective.

MOO problems can be addressed via a priori, a posteriori and interactive methods. In a priori methods, multiple objectives are aggregated into one single objective. Advantages of a priori methods are decreased problem complexity (SOO algorithms suffice for solving the problem) and low computational costs. Disadvantages are the need to decide on objective importance before optimization which requires strong domain knowledge, the need to run the optimization multiple times to be able to find Pareto optimal solutions, it is not guaranteed to find uniformly

distributed Pareto optimal solutions and most importantly, non-convex regions of the Pareto front cannot be obtained using this method [18].

The authors in [12] use an a priori method in which the minimization of the number of setups at the press brake and the minimization of make span of a metal sheet are aggregated into one objective function. To reduce problem complexity, only a cutting and bending process where both work centers comprise of one machine are considered. The simplified problem is then modeled using LP. The LP results in a grouping of parts to sheets, not a placement of parts on sheets. In order to sequence the metal sheets for cutting such that setups are minimized at the press brake, a TSP is used. By applying branch-and-bound techniques, the optimal solution for a small use case is found. If the problem size is increased, the proposed algorithm is not capable of finding a solution within a acceptable time frame.

The SOO problem previously defined in [15] is extended to a MOO problem in [19]. The objective function is altered such that both makespan (with an emphasis on generating a production planning using a minimal number of metal sheets) and flowtime at the bending work center are minimized. To solve this MOO problem, a bicriteria optimisation is proposed to determine the Pareto set. Because calculating the Pareto set is computationally expensive, it is suggested to approximate it by using a budget approach. In this approach, one criterion is minimised while the other is bound to a budget. The proposed heuristic method is capable of minimizing make span with a secondary criterion, in this case total flow time. Results show that a reduction in total flow time is possible without deteriorating make span. With a small deterioration of make span, even larger improvements in flow time can be realised.

In a posteriori methods, a MOO algorithm is employed to find the best trade-off between the objectives at hand. To determine if one solution dominates another, a new operator called Pareto optimal dominance is introduced. According to this operator, one solution dominates another if it shows equal objective values on all objectives and is at least better in one of the objectives.

An a posteriori method using bottom left algorithm based batching at the cutting work center and dispatching rule based scheduling for the other work centers is proposed in [20]. Both the batching and scheduling problem are formulated as LP. The objective function of the batching problem minimizes waste material, the objective function of the scheduling problem minimizes tardiness. To reduce problem complexity, the authors assume constant material thickness. Although test results indicate that the proposed method can derive desirable solutions, the applicability to real world application is limited by the assumption of constant material thickness.

In [21] and [22] the authors propose improvements to their research in [20] in order to increase solution quality. Regarding the batching problem, a GA is proposed to revise the operational sequence of the cutting process. Scheduling is executed first to determine a cutting layout in the batching stage. After the batching stage is finished, the initial schedule is revised based on the obtained cutting layouts. The revision is executed by means of a GA in combination with a appropriate dispatching rule. To enhance scheduling ability of the algorithm, a heuristic approach is proposed. The heuristic comprises a local search (VNS) to update the initial schedule, which is decided by a Earliest Due Date (EDD) based dispatching rule, and manage it in terms of the criteria referring to the bottleneck process.

In a more recent contribution [13] an a posteriori method which jointly minimizing tardiness and material waste for medium-sized, offline sheet metal jobshop instances using a RL approach, specifically AlphaGo Zero (AZ), is proposed. In this research, setup times, machine/worker availability and transportation times between workstations are not considered. Furthermore, the two-dimensional packing problem is flattened into one dimension. The cutting constraint states that the summed area of all parts batched on a metal sheet must not exceed the total sheet area. A single player AZ version is pretrained using supervised learning on schedules generated

by a heuristic, EDD in this case. The schedule is then fine-tuned using RL and evaluated through comparison with a heuristic baseline and Monte Carlo Tree Search. The authors conclude that the proposed AZ outperforms the other two approaches. The used testing instance is fairly large compared to other researches, however not large enough to be representable for real life situations.

This paper can be framed among those works using an a posteriori method to solve the JSSP considering batches. Compared to an a priori method, an a posteriori method poses several benefits in finding the Pareto set. Deciding on objective importance is done after optimization, requiring less domain knowledge. Furthermore, an a posteriori method is guaranteed to find a uniformly distributed Pareto set. This is beneficial during the decision making process, as a higher amount of solutions is presented to the decision maker enabling a better trade-off between conflicting objectives. Most importantly, non-convex regions of the Pareto set can be obtained when using an a posteriori method, resulting in higher solution quality [18].

This study further develops the work in [22] by extending the model proposed in this work. With respect to the proposed model, the constraints of homogeneous material thickness, rectangular part area, no setup times, predefined machine assignment and predefined cutting time per batch are relaxed. However, the batching problem is simplified by omitting placement of parts within batches. Furthermore, the work in [13] is further developed in this study by adding setup times to the proposed model and by redefining the one dimensional bin packing problem (1D-BPP) to enhance batch feasibility. In accordance with both [22] and [13], this study does not consider machine availability, worker availability and transportation times.

The main contribution with respect to the contributions mentioned above, is the emphasis on Pareto set generation during solution generation. Compared to a solution generation method resulting in one optimal solution, used by both [22] and [13], a Pareto set enables a decision making process in which the trade-off between conflicting objectives can be decided upon after optimization. A Pareto set provides insights in how different production schedules affect production objectives and enables to substantiate a decision for a certain trade-off. Furthermore, it contributes to the explainability of the optimization algorithm by visually representing the outcome of the algorithm.

Determining the Pareto set is a NP-hard problem [23]. Solving a NP hard problem using mathematical programming (MP) techniques is possible for small problem instances but becomes more computational expensive when instance size increases. To be able to solve large MOO problem instances in a reasonable time frame, metaheuristics are used. Metaheuristics are designed such that the complex problem at hand can be solved faster and more efficient by sacrificing optimality, accuracy, precision, or completeness for speed [24] [25]. Metaheuristic approaches can be classified according to the number of solutions that are evolved at each stage of the algorithm, single-solution based metaheuristics and population-based metaheuristics [26].

In [27] the difference between single-solution based metaheuristics and population-based metaheuristics is described and an overview of the main domains amongst these two classes is provided. Single-solution based metaheuristics, also called trajectory methods, start with a single initial solution and move away from it, describing a trajectory in the search space. Single-solution based MOO metaheuristics are composed of the simulated annealing based algorithm AMOSA [28], the tabu search based algorithms MOTS and PRMOTS [29], the greedy randomized adaptive search procedure (GRASP) algorithm MOG [30], the variable neighborhood search algorithms MO-RVNS, MO-VND and MO-GVNS [31], the guided local search based algorithm MOGLS [32] and finally the iterated local search algorithm PILS [33].

Population-based metaheuristics deal with a set, or a population, of solutions rather than a single solution. There are mainly two fields of population-based metaheuristics, Swarm Intel-

ligence (SI) and Evolutionary Computation (EC). SI algorithms are inspired by (social) interaction between animals, EC algorithms by Darwinian evolution. For each of the metaheuristic fields discussed next, MOO algorithms within the field are mentioned. SI contains the domains of ant colony optimization (BicriterionAnt [34], Pareto ACO [35]), particle swarm optimization (MOPSO [36], AMOPSO [37], OMOPSO [38]), bacterial foraging optimization (MBFO [39]), bee colony optimization (MOABC [40], MOABC [41], eMOABC [42]), artificial immune systems (VIS [43], MOBAIS [44], EMOIA [45]) and bio geography-based optimization (MOBBO [46]). MOO EC contains the domains of genetic algorithms (SPEA2 [47], NSGA-II [48], NSGA-III [49]), evolution strategies (M-PAES [50], ESP [51], MO-CMA-ES [52]), evolutionary programming (MOEP [53]), genetic programming (multiple problem specific MOGP algorithms exist in literature [54]) and differential evolution (MODE [55], AMODE [56], JADE [57], jDE [58], SHADE [59], L-SHADE [60], PWI-based L-SHADE [61], HyDE [62], WDE [63], IMODE [64], GDE3 [65]).

When analysing the different metaheuristic fields, the field of differential evolution (DE) shows a higher density of recent research dedicated on improving multi objective DE algorithms compared to the other fields. This is a first indication of the potential of this field. Furthermore, literature shows that multi objective DE has a higher convergence rate and efficient global search capability compared to other multi objective evolutionary algorithms (EAs).

In [66] and [67] the authors compare and evaluate the performance of DE and PSO. This is done on test sets of respectively 34 benchmark problems and 8 benchmark functions. Results show that DE generally performs better than PSO in term of solution accuracy and robustness in most test cases.

In a more recent contribution [68] a variant of DE called DEEP is proposed. The proposed algorithm is used to optimize wind farm layout. Algorithm performance is compared with the greedy method called turbine distribution algorithm (TDA) [69] and five evolutionary algorithms: CMA-ES [70], MSO [71] and CLPSO [72], JADE and SHADE. Experimental results show that the proposed algorithm generally achieves the highest solution accuracy with the fastest convergence speed in a robust manner. Results also indicate that the proposed algorithm has low parameter sensitivity.

The performance of DE, PSO, QPSO [73], DSA [74] and mixed integer nonlinear programs (MINLP) is compared and evaluated in [75]. Results indicate that DE strategies can find better solutions compared to the other evaluated evolutionary algorithms and DE strategies can find near-optimal solutions in a acceptable time frame compared to a MINLP approach. However, unlike [68], the DE strategies used are very sensitive to the setting of their parameters.

In a consecutive research [62], the authors propose a new DE algorithm called HyDE and compare and evaluate its performance with standard DE, JADE and jDE [58]. Results show that the proposed algorithm outperforms the other algorithms both in terms of solution quality and convergence capability.

Recent contributions proposing new variants of DE are [61], [63] and [64]. The proposed algorithms are called PWI-based L-SHADE, WDE and IMODE. PWI-based L-SHADE is tested on 60 artificial benchmark problems from IEEE CEC'2014 competition and IEEE CEC'2017 competition and on 22 real-world problems from IEEE CEC'2011 competition. Algorithm performance is compared with 16 different metaheuristics which are all outperformed by the proposed algorithm. Only 2 of 16 metaheuristics may be considered competitive on some problem sets. WDE is tested on IEEE CEC'2013 competition problems and its performance is compared with CS [76], ABC [77], JADE and BSA [78]. Results show that WDE significantly outperforms the other algorithms. Lastly, IMODE performance is compared with EBOwithC-MAR [79], HSES [80], LSHADE-cnEpSin [81] and LSHADE-SPACMA [82]. For testing, 10 problems from IEEE CEC'2020 competition on single objective bound constrained optimization are used. Results show that IMODE statistically outperforms the other algorithms. The success of multi objective DE and its variants is underlined by winning several IEEE competitions. For example

in [83], a GECCO 2015 competition on optimizing wind farm layout, the proposed DE algorithm 3s-MDE outperforms Evolution Strategy (ES) algorithm CMA-ES [84], Sequence-based Selection Hyper-Heuristic (SSHH) [85], Goldman Method (GM) and a GA.

In the present research, a JSSP considering batches containing two objectives is solved in order to find a production schedule which minimizes both waste material incurred during batching and total tardiness of jobs. Both the batching and scheduling problem will be modeled using an a posteriori method. This study further develops the work in [22] and [13] with the inclusion of setup times within the model and the emphasis on Pareto set generation by using the metaheuristic DE for solution generation. Amongst MOO metaheuristics, DE shows better performance regarding Pareto set quality and generation on several benchmark problems and is therefore selected. The use of DE for solution generation under an a posteriori method within a JSSP considering batches is a novelty.

3 Problem description

This research studies a JSSP considering batches in which a 1D-BPP is combined with a FJSP. This results in a MOO problem containing two objectives. The objective posed by the 1D-BPP is to minimize the waste material per batch b in set B. Because batches are defined when solving the 1D-BBP, we rephrase minimizing waste material to minimizing the number of batches b in set B. The objective posed by the FJSP is to minimize total tardiness for all jobs j in set J.

Jobshop work orders are controlled by a higher level planning within the company which is based on customer demand, resulting in dynamic order arrivals. After arrival to the system, the set of work orders is treated as a set of jobs $J = \{1, 2, ..., |J|\}$. A job j consists of one or more similar sheet metal parts. The part(s) within a job determine the jobs area a_j, material r_j, thickness q_j and set of operations $O_j = \{(j, o), o = 1, 2, ..., |O_j|\}$ needed to produce the job, with $O = \cup_{j \in J} O_j$ being the total set of operations available in the jobshop. The notation (j, o) can be interpreted as the oth operation of job j. The processing time of operation o at machine m is denoted as p_{jom}. Processing times for all operations except cutting and picking are known when a job enters the system. The processing times of cutting and picking need to be calculated after batch definition, so during schedule computation, and will be denoted with p_{bom}. All operations present in O_j are executed sequentially, where operation $o + 1$ can only start when operation o is finished. Each job j has a due date d_j at which all operations in O_j should be finished.

Each operation $o \in O$ can be processed by a machine within set $M_o = \{1, 2, ..., |M_o|\}$. Each operation $(j, o) \in O_j$ can only be processed by one machine $m \in M_o$ at a time. The operations of laser cutting and picking will be denoted with respectively $o = 1$ and $o = 2$. The machines within sets M_1 and M_2 execute batch processing. Each batch b needs to be assigned to a machine in M_1 and M_2 and each machine in M_1 and M_2 can process one batch b at a time. For $o \in O, o > 2$, the machines sets M_o operate in a flexible jobshop environment, meaning that a machine needs to be assigned to every operation (j, o) and that each machine m can only process one operation of one job (j, o) at a time. The relation between jobs, batches and the different sets of machines is visually represented in Figure 1. The set of jobs at the left hand side contains jobs to be processed within the jobshop and thus forms the input to the system. The set of jobs on the right hand side contains jobs which have finished processing within the jobshop and thus forms the output of the system. The arrows indicate the flow of jobs through the jobshop.

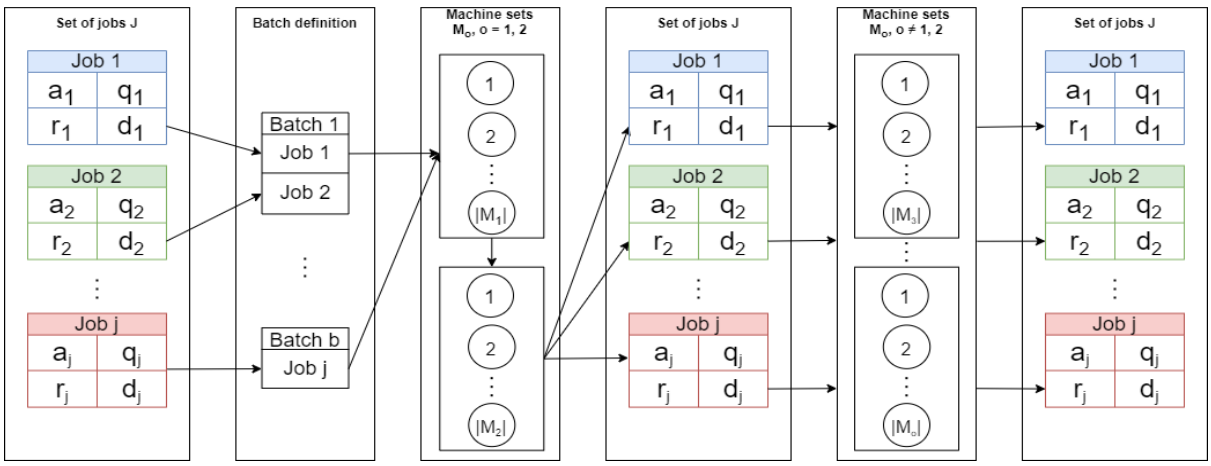


Figure 1: Jobshop process overview

3.1 Definition and processing of batches

In order to create a feasible batch b, a 2D-BPP needs to be solved. This problem consists of packing a set of two-dimensional rectangular items into the minimum number of two-dimensional rectangular bins without overlap. All bins have identical width and height whereas each item has a specific width and height. The 2D-BPP is a NP-hard problem [14]. In order to reduce problem complexity, the 2D-BPP is flattened into a one dimensional problem by omitting placement of parts. In order to maintain batch feasibility under this assumption, the constant a^b is introduced. Here, a^b is a certain fraction of the available sheet area. The summed area of all jobs in a batch $\sum_{j \in b} a_j$ must be smaller than the constant a^b. To understand why a^b is a fraction of the available sheet area, we first have to understand how the placement of parts effects solving a 2D-BPP in a jobshop environment. We will explain this using an example of a batch, which is depicted in Figure 2. In the figure, the parts of two jobs are placed within a batch. The white space is unused sheet area. The white space between parts on the left hand side and in the middle of the batch is left empty to account for cutting seams. On the right hand side of the batch we see larger white spaces. This area remained unused because the set of jobs either contained one or more jobs which could be placed in this area based on a_j but could not be placed without overlap, or the set of jobs simply did not contain a job with a area a_j small enough to fit in the available space. By omitting placement of parts, and thus not accounting for cutting seams or placement of parts without overlap, we have to compensate for these during batch definition to create feasible batches under the simplification. To do so, the constant a^b is a fraction of the sheet area used in real life situations.

Furthermore, as batched jobs are cut out of one metal sheet with uniform material and thickness, jobs can only be batched if they have the same material $r_j = \{1, 2, ..., |r_j|\}$ and thickness $q_j = \{1, 2, ..., |q_j|\}$. To indicate whether job j is assigned to batch b, the decision variable x_{jb} is introduced. This variable is 1 if job j is assigned to batch b and 0 otherwise.

The objective when solving the 1D-BPP is to maximize the number of jobs per batch as to minimize the number of batches b in set B. As the number of batches b in set B is not known a priori, it is defined as its upper bound, being $|J|$. Under this assumption, each job $j \in J$ is assigned to a unique batch. To indicate whether batch $b \in B$ is used, the decision variable x_b is introduced. This variable is 1 if batch b is used and 0 otherwise.

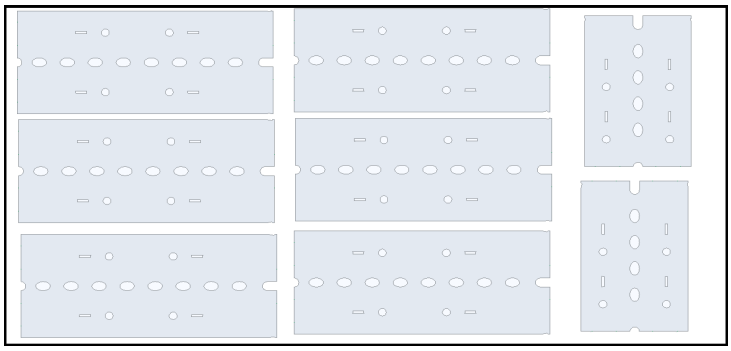


Figure 2: Two orders nested on a sheet metal base plate with material thickness 4mm

To indicate whether operation o of batch b is assigned to machine m in set M_o, the decision variable x_{bom} is introduced. This variable is 1 if operation (b, o) is assigned to machine m and 0 otherwise. The processing sequence per machine within sets M_1 and M_2 is defined by decision variable t^s_{bom}. This variable, using integer values, indicates the starting time of operation $(b, o)\ \forall\ o \in O_j \mid o = (1, 2)$ at machine m and will be 0 for all machines which are not selected for the operation. Machines within sets M_1 and M_2 use serial batch processing, implying that the batch processing time p_{bom} for the two sets is defined as respectively the sum of the individual cutting processing times of all jobs within a batch or the sum of the individual picking times of

all jobs within a batch [86]. Once p_{bom} is determined for both machine sets, a batch is ready to be processed. Batches awaiting processing need to be assigned to machines within set M_1 and the processing sequence per machine needs to be determined. Machines within this set incur sequence independent setup times, namely the setup time $s_{om},\ m \in M_1,\ o = 1$ depends neither on the current batch, nor the preceding batch [87].

Once a batch has been processed by one of the machines in set M_1, it is ready to be processed by one of the machines within set M_2. Again, batches awaiting processing need to be assigned to machines within set M_2 and the processing sequence per machine needs to be determined. Between batches, a sequence independent setup time is incurred. During processing at a machine within set M_2, jobs contained in a batch are separated again for further processing in the jobshop.

3.2 Processing of jobs

For all sets $M_o,\ o > 2$, jobs awaiting processing need to be assigned to a machine and the processing sequence per machine needs to be determined. The objective amongst these machine sets is to minimize total tardiness of all jobs. When job j completed processing in the jobshop, it leaves the system.

To indicate whether operation (j, o) is assigned to machine m in set M_o, the decision variable x_{jom} is introduced. This variable is 1 if operation (j, o) is assigned to machine m and 0 otherwise. For machines in machine sets processing jobs, the decision variable t^s_{jom} is to determine the processing sequence of jobs. This variable, using integer values, indicates the starting time of operation $(j, o)\ \forall\ o \in O_j \mid o > 2$ at machine m and will be 0 for all machines which are not selected for the operation. The completion time of an arbitrary operation o at an arbitrary machine m can be calculated by adding the sum of setup time s_{om} and processing time p_{bom} or p_{jom} to the corresponding starting time of the considered operation. The processing sequence per machine can be derived from respectively t^s_{bom} or t^s_{jom} using the logic that for each $m \in M_o$, the sorted non-zero values of t^s_{bom} or t^s_{jom} indicate the processing sequence.

4 Mathematical formulation

The JSSP considering batches is mathematically formulated in this section. Firstly, the objectives are defined. Both objective functions are subjected to several constraints, of which constraints 3 to 9 constrain the processing of jobs, constraints 10 to 22 constrain the definition and processing of batches and constraint 23 forms a link between the processing of jobs and batches. The mathematical program is a integer nonlinear program (INLP). It is integer because all decision variables can only take integer values and nonlinear because two decision variables are multiplied in objective function 2. The objective functions are defined below.

$$\min f(x) = \sum_{b \in B} x_b \tag{1}$$

$$\min g(t) = \sum_{j \in J} \max(0, (t^s_{jom} + p_{jom} + s_{om}) * x_{jom} - d_j) \ \forall \ m \in M_o, \ o = |O_j| \tag{2}$$

Objective function $f(x)$ represents the number of used batches b in set B. As the batches in set B are defined within the mathematical program, the size of set B is not known a priori. Therefore, the maximum size of set B is defined as $|J|$. In order to minimize the number of used batches in set B, objective function $f(x)$ is defined. The second objective function $g(t)$ represents the total tardiness of all jobs in set J. Similarly to the first objective function, this function is minimized as well. This function is non-linear, because the decision variables t_{jom} and x_{jom} are multiplied in this function. Next, constraints regarding the processing of jobs are formulated and explained.

$$t^s_{jom} \geq t^s_{j,o-1,m} + p_{j,o-1,m} + s_{o-1,m} \ \forall \ j \in J, \ o \in O_j \mid o > 3, \ m \in M_o \mid o > 2 \tag{3}$$

$$\sum_{m \in M_o} x_{jom} = 1 \ \forall \ j \in J, \ o \in O_j \mid o > 2 \tag{4}$$

$$n_m = \sum_{j \in J} x_{jom} \ \forall \ m \in M_o \mid o > 2, \ o \in O \mid o > 2 \tag{5}$$

$$\sum_{j \in J} \sum_{k \in J} x_{jkm} = n_m - 1 \ \forall \ m \in M_o \mid o > 2 \tag{6}$$

$$\sum_{j \in J} x_{jkm} \leq 1 * x_{jom} \ \forall \ j \in J, \ k \in J, \ o \in O \mid o > 2, \ m \in M_o \mid o > 2 \tag{7}$$

$$\sum_{k \in J} x_{jkm} \leq 1 * x_{jom} \ \forall \ j \in J, \ o \in O \mid o > 2, \ m \in M_o \mid o > 2 \tag{8}$$

$$t^s_{jom} + p_{jom} + s_{om} \leq t^s_{kom} + M * (1 - x_{jkm}) \ \forall \ j \in J, \ k \in J, \ o \in O_j \mid o > 2, \ m \in M_o \mid o > 2 \tag{9}$$

Constraint 3 ensures that all operations $o > 2$ in set O_j are processed sequentially. Furthermore, this constraint ensures that operation o cannot start processing until the preceding operation $o - 1$ has finished processing. To process operation o of job j, it needs to be assigned to one single machine in set M_o. This is defined in constraint 4. The variable parameter n_m is defined in constraint 5 and indicates the number of jobs which need to be processes by machine m. In constraints 6 to 8, the decision variable x_{jkm} is defined. This variable is 1 if job j is processed before job k at machine m. Using x_{jkm}, constraint 9 defines that each machine m in set M_o can only process one job j at a time. Because the constraint should only be active when job j is processed before job k at machine m, M is introduced. M is a large number, enforcing

the constraint to only be active when $x_{jkm} = 1$. Next, constraints regarding the definition and processing of batches are formulated and explained.

$$\sum_{b \in B} x_{jb} = 1 \ \forall \ j \in J \tag{10}$$

$$\sum_{j \in J} a_j * x_{jb} \leq a^b * x_b \ \forall \ b \in B \tag{11}$$

$$\frac{\sum_{j \in J} r_j * x_{jb}}{r_j} \leq \sum_{j \in J} x_{jb} + M(1 - x_{jb}) \ \forall \ j \in J, \ b \in B \tag{12}$$

$$\frac{\sum_{j \in J} r_j * x_{jb}}{r_j} \geq \sum_{j \in J} x_{jb} - M(1 - x_{jb}) \ \forall \ j \in J, \ b \in B \tag{13}$$

$$\frac{\sum_{j \in J} q_j * x_{jb}}{q_j} \leq \sum_{j \in J} x_{jb} + M(1 - x_{jb}) \ \forall \ j \in J, \ b \in B \tag{14}$$

$$\frac{\sum_{j \in J} q_j * x_{jb}}{q_j} \geq \sum_{j \in J} x_{jb} - M(1 - x_{jb}) \ \forall \ j \in J, \ b \in B \tag{15}$$

$$p_{bom} = \sum_{j \in J} p_{jom} * x_{jb} \ \forall \ b \in B, \ o \in O_j \mid o = (1,2), \ m \in M_o \mid o = (1,2) \tag{16}$$

$$t^s_{bom} \geq (t^s_{b,o-1,m} + p_{b,o-1,m} + s_{o-1,m}) * x_b \ \forall \ b \in B, \ o \in O_j \mid o = 2, \ m \in M_o \mid o = 2 \tag{17}$$

$$\sum_{m \in M_o} x_{bom} = 1 \ \forall \ b \in B, \ o \in O_j \mid o = (1,2) \tag{18}$$

$$\sum_{b \in B} \sum_{h \in B} x_{bhm} = \sum_{b \in B} x_{bom} - 1 \ \forall \ o \in O \mid o = (1,2), \ m \in M_o \mid o = (1,2) \tag{19}$$

$$\sum_{b \in B} x_{bhm} \leq 1 * x_{bom} \ \forall \ b \in B, \ h \in B, \ o \in O \mid o = (1,2), \ m \in M_o \mid o = (1,2) \tag{20}$$

$$\sum_{h \in B} x_{bhm} \leq 1 * x_{bom} \ \forall \ b \in B, \ o \in O \mid o = (1,2), \ m \in M_o \mid o = (1,2) \tag{21}$$

$$t^s_{bom} + p_{bom} + s_{om} \leq t^s_{hom} + M * (1 - x_{bhm}) \ \forall \ b \in B, \ h \in B, \ o \in O \mid o = (1,2), \ m \in M_o \mid o = (1,2) \tag{22}$$

Constraint 10 defines that each job needs to be assigned to one batch b. Constraints 11 to 15 constrain the definition of feasible batches. Firstly, the summed area of jobs assigned to batch b is constrained to be less than or equal to the batch area a^b, where a^b is defined as the total batch area multiplied with a safety factor to enhance batch feasibility under the simplification of a 1D-BPP. Because a batch equals a sheet metal plate which has uniform material and thickness, all jobs assigned to batch b should have equal material and thickness properties. Both requirements are defined in constraints 12 to 15. Because all four constraints should only be active when $x_{jb} = 1$, indicating that job j is assigned to batch b, M is used. M enforces the four constraints to only be active when $x_{jb} = 1$.

As batch definition is done during schedule computation, batch processing times for batch processing operations need to be constrained. This is done in constraints 16. Because batch

processing machines make use of serial batch processing, processing times can be defined as the sum of the processing times p_{jom}, given that o is a batch processing operation, of all jobs j assigned to batch b. In constraint 17, process sequence of batch processing operations $o = 1$ (cutting) and $o = 2$ (picking) is defined. Furthermore, this constraint ensures that operation o cannot start processing until the preceding operation $o - 1$ has finished processing. To process operation o of batch b, it needs to be assigned to one single machine in set M_o. This is defined in constraint 18. In constraints 19 to 21, the decision variable x_{bhm} is defined. This variable is 1 if batch b is processed before batch h at machine m. The last batching constraint, constraint 22, ensures that each machine m in sets M_1 and M_2 can only process one batch b at a time. To do so, the constraint uses x_{bhm} in combination with M to activate the constraint when $x_{bhm} = 1$. To link the processing of jobs and batches together, constraint 23 is defined.

$$t^s_{bom_2}+p_{bom_2}+s_{om_2} \leq t^s_{j,o+1,m_1}+M*(1-x_{jb}) \; \forall \; j \in J, \; b \in B, \; o = 2, \; m_1 \in M_o \mid o = 2, \; m_2 \in M_o \mid o = 3 \tag{23}$$

In constraint 23, the starting time of the first operation o of a job following batch operations is constrained to be larger or equal to the time at which the last batching operation $o = 2$ is finished. M is used to activate the constraint only when job j was assigned to batch b. Lastly, a non-negativity constraint for all decision variables is defined in 24.

$$t^s_{jom} \geq 0, \; x_{jom} \in \{0,1\}, \; x_{jkm} \in \{0,1\}, \; x_b \in \{0,1\}, \; x_{jb} \in \{0,1\}, \; t^s_{bom} \geq 0, \; x_{bom} \in \{0,1\}, \; x_{bhm} \in \{0,1\} \tag{24}$$

4.1 Linearize model

As stated in the previous section, objective function $g(t)$ causes the model to be non-linear. In order for a solver, such as Gurobi, to solve the model, $g(t)$ should be linearized. To do so, both the max statement and the multiplication of decision variables need to be rewritten. The new linear objective function is given in Equation 25. Constraints necessary for the linearization are given in constraint 26 to 30.

$$\min g(v) = \sum_{j \in J} v_j \tag{25}$$

Subjected to;

$$v_j \geq 0 \; \forall \; j \in J \tag{26}$$

$$v_j \geq (p_{jom} + s_{om}) * x_{jom} + z_j - d_j \; \forall \; j \in J, \; o \in O_j \mid o = |O_j|, \; m \in M_o \mid o = |O_j| \tag{27}$$

$$z_j \leq t^s_{jom} + p_{jom} + s_{om} \; \forall \; j \in J, \; o \in O_j \mid o = |O_j|, \; m \in M_o \mid o = |O_j| \tag{28}$$

$$z_j \geq t^s_{jom} + p_{jom} + s_{om} - M * (1 - x_{jom}) \; \forall \; j \in J, \; o \in O_j \mid o = |O_j|, \; m \in M_o \mid o = |O_j| \tag{29}$$

$$z_j \leq M * x_{jom} \; \forall \; j \in J, \; o \in O_j \mid o = |O_j|, \; m \in M_o \mid o = |O_j| \tag{30}$$

All variables used within the mathematical formulation are described below.

Decision variables

t^s_{jom}	: Starting time of setup of operation (j, o) on machine m
t^s_{bom}	: Starting time of setup of operation (b, o) on machine m
x_b	: 1 if batch b is used, 0 otherwise
x_{jb}	: 1 if job j is assigned to batch b, 0 otherwise
x_{jkm}	: 1 if job j is processed before job k at machine m, 0 otherwise
x_{jom}	: 1 if machine m is selected for operation (j, o), 0 otherwise
x_{bom}	: 1 if machine m is selected for operation (b, o), 0 otherwise
x_{bhm}	: 1 if batch b is processed before batch h at machine m, 0 otherwise

Sets

J	: Set of jobs
O	: Set of operations
O_j	: Set of operations needed to produce job j
B	: Set of batches
M_o	: Set of machines on which operation o can be processed

Variable parameters

n_m	: Number of jobs produced at machine m
p_{bom}	: Processing time of operation (b, o) at machine m

Constant parameters

p_{jom}	: Production time of operation (j, o) at machine m
s_{om}	: Setup time of operation o at machine m
d_j	: Due date of job j
a_j	: Area of job j
q_j	: Thickness of job j
r_j	: Material of job j
a^b	: Batch area
M	: Large number

5 Solution approach

To solve the mathematical problem we propose to use a variant of the metaheuristic DE to find an approximation of the non-dominated Pareto set. In particular, we propose to use Generalized Differential Evolution 3 (GDE3) [65]. In [79], the authors state that GDE3 received a winning entry nomination in the CEC 2007 competition and was ranked among the top 5 best performing algorithms in the CEC 2009 competition. Furthermore, GDE3 is ranked amongst the best performing algorithms in several studies by [88], [89] and [90]. Next to proven algorithm performance, GDE3 is available in the Python framework for DE called Pymoode [91]. Pymoode is based on Pymoo [92], a Python framework for MOO. GDE3 is an extension of DE purposed for global optimization with an arbitrary number of objectives and constraints. The algorithm combines DE mutation and crossover operators with an improved NSGA-II survival strategy. The improved survival strategy ensures a better distributed solution set. Furthermore, compared to earlier versions of GDE, GDE3 is less sensitive to the selection of control parameter settings [65].

Comparable to a typical EA, GDE3 starts with some random initial population, which is then improved using selection, mutation and crossover operations. Several control parameters can be used to control these operations. The GDE3 algorithm, together with its control parameters are discussed in subsection 5.1. How to tune the different control parameters for optimal algorithm performance will be discussed in subsection 5.2. Next, algorithm performance measures will be discussed in subsection 5.3. The mathematical model discussed in section 4 was implemented in the Python framework Pymoode. The implementation is discussed in subsection 5.4. The JSSP considering batches has differently scaled objectives. Therefore, decision space normalization is discussed in subsection 5.5.

5.1 GDE3 and its control parameters

The GDE3 algorithm is introduced in [65]. In each generation of the algorithm, GDE3 goes through each individual X in the population and creates a corresponding candidate individual. Within the GDE3 implementation in Python, X is represented by a vector containing the decision variables t^s_{jom} and x_{jb}. The amount of values for t^s_{jom} present in X equals the number of jobs multiplied with the number of operations. This translates to a starting time for each job at each operation. The amount of values for x_{jb} present in X equals the number of jobs squared. This translates to a separate batch for each job. Note that not all batches should be used by the algorithm. Furthermore, it should be noted that not all decision variables mentioned in section 4 are represented in X. In Python, t^s_{jom} and x_{jb} are sufficient to define the objective functions and constraints. Therefore, it is possible to leave out the other decision variables and limit the dimensions of X.

Creation of a candidate individual is done using the most common DE version, DE/rand/1/bin. In DE/rand/1/bin, three individuals a, b and c are randomly selected from the population excluding initial individual X. Using these three individuals, a mutation vector is constructed using the function $y = a + F * (b - c)$, where $F \in (0, 1+]$ [93] is a scaling factor for mutation. F controls the speed and robustness of the search and as such, F determines the trade-off between exploration and exploitation. A low value for F will increase the convergence rate while increasing the risk of finding a local optimum. Once the mutation vector is constructed, the candidate individual is created using crossover between the initial individual x and the mutation vector y. Each chromosome of the candidate individual takes the value of the corresponding chromosome within the mutation vector with probability $CR \in [0, 1]$ [93]. CR controls the degree of mutation and therefore the convergence speed of the algorithm.

Next to DE/rand/1/bin, any other DE strategy can be used within the GDE3 algorithm [93]. Pymoode also facilitates implementing different strategies. As the main focus of this research is on applying GDE3 on a JSSP considering batches, we will focus on the most common DE

version and devote the testing of different versions to future research.

After creation, the candidate individual is evaluated using the objective functions, based on which selection is performed. This is the operation where GDE3 differs from DE. In DE, the candidate solution is selected and replaces the initial individual X in the population if the candidate solution scores better than X. In GDE3, three rules are used to perform selection. Firstly, when both individuals are infeasible, the candidate individual is selected if it weakly dominates X in the constraint violation space, otherwise X is selected. The constraint violation space is the set of all possible constraint violations. Secondly, in case of one feasible and one infeasible individual, the feasible individual is selected. Lastly, in case of two feasible individuals, the candidate individual is selected if it weakly dominates X in the objective space, otherwise X is selected. If neither of the two individuals dominates the other, both individuals are selected for the next generation. In the last case, the population will increase after a generation. If so, it will be decreased to the original size based on a selection approach used in NSGA-II.

Next to F and CR, the last control parameter to be set is the number of individuals N within the population. In order to perform mutation, crossover and selection operations, at least 4 individuals are needed, thus $N \in [4, \infty]$. In [93], the authors recommend to set N between $5D$ and $30D$ for a MOO problem with conflicting objectives, where D represents the dimension of an individual X. In case of the JSSP considering batches, $D = n * o + n^2$ where n equals the number of jobs and o the number of operations.

5.2 Parameter optimization

In [94], the authors conclude that for different optimization problems different parameter settings for F and CR are needed to obtain good solution quality. So, parameter optimization is problem depended. Because GDE3 has never been used to optimize a JSSP considering batches, a parameter optimization should be performed. Amongst EAs, parameter optimization can be divided in two sub classes, parameter tuning and parameter control. In parameter tuning, good values for the parameters are determined before the algorithm is executed and will remain fixed during a run. In parameter control, a run is started with a initial set of parameter values which are then changed during a run to enhance algorithm performance [95]. As the main focus of this research is on applying GDE3 on a JSSP considering batches, we will focus on parameter tuning and devote parameter control to future research.

Amongst parameter tuning methods, the authors in [96] distinguish between black box optimization, multi-fidelity optimization and algorithmic approaches. Black box optimization contains the model-free methods grid search and random search and the Bayesian optimization method. In the model free methods, a finite set of values is specified for each parameter. Where grid search evaluates the Cartesian product of these sets, random search samples from the sets at random. From the two methods, random search is computationally less expensive and is more likely to find optimal parameter settings [97]. In Bayesian optimization method, a surrogate model is formed based on sampled points from the target function. Based on the surrogate model promising points are identified. An acquisition function then determines the utility of different promising points, trading off exploration and exploitation and will update the surrogate model accordingly. Bayesian optimization is primarily used when the objective function is expensive to evaluate. In multi-fidelity optimization, low fidelity approximations of the actual loss function are used to significantly lower optimization run time by only slightly giving in on optimization performance. Multi-fidelity methods can make use of a subset of the data, training only for a few iterations or run on a subset of features. Finally, in algorithmic approaches, an EA is used to optimize the parameters of another EA. The authors in [98] give an extensive overview of available algorithmic approaches.

In this research, we propose to use a combination of the tuning methods multi-fidelity optimization and grid search. In particular, we propose to use a subset of the data to decrease computational costs in combination with a multi stage grid search. The exact size of the subset

is calculated in section 6. In the course of the grid search, the grid will be updated based on results of former stages. The bounds of the variables are set according to the intervals proposed by [93]. So $N \in [5D, 30D]$, $CR \in [0, 1]$ and $F \in (0, 2]$.

Similar to other EAs, GDE3 has a stochastic nature. Therefore, multiple runs of the parameter optimization are needed to obtain statistically significant results. Executing multiple runs enables determining of two types of robustness, being robustness to change in parameter settings and robustness of change in random seed [98].

5.3 Performance measures

Algorithm performance measures can be divided in two sub classes, measures for the case in which the true Pareto front has been derived analytically or an approximation can be made and measures for the case in which the true Pareto front is not known. In case of this study, the Pareto front is not known and therefore we will focus on measures belonging to this class. Measures within this class are convergence speed with regards to population feasibility, hypervolume indicator, running metric and robustness. All these performance measures will be used to measure algorithm performance in section 6.

Convergence speed with regards to population feasibility is a method to check the speed at which an EA is able to find feasible solutions within the solution space. A solution is deemed feasible if all constraints are satisfied.

The hypervolume indicator was first proposed by [99] as a method to evaluate Pareto fronts produced by EAs. It describes the size of the objective value space which is covered by the set of non-dominated solutions within the Pareto front. Hypervolume indicator is integrated in Pymoo and, according to the available documentation, it can be calculated efficiently for 2 to 3 objectives. In case of more objectives, computations become expensive.

The running metric is recently proposed by [100] and is used to analyse the convergence of an EA. Different from the other metrics mentioned in this section, the running metric can be applied at any time during algorithm execution to measure algorithm performance. Like hypervolume, running metric is integrated in Pymoo.

As discussed in the former section, GDE3 has a stochastic nature. Therefore, multiple runs with the optimal parameter setting are needed to obtain statistically significant results for the JSSP considering batches. This enables determining robustness to change in random seed.

5.4 GDE3 implementation

The objective functions and constraints mentioned in section 4 are defined within the Pymoode framework in Python. In subsection 5.1, we already discussed how vector X represents the decision variables within the Python implementation. When an individual X needs to be evaluated by the model, it is converted to two matrices. A matrix $\mathbf{T}$ with dimensions $|J| \times |O|$, containing all t^s_{jom} values present in X, and a matrix $\mathbf{D}$ with dimensions $|J| \times |J|$, containing all x_{jb} values present in X.

To test the implementation, a dummy data set containing 2 jobs and with known optima is introduced. Extensive testing showed that the algorithm is able to satisfy all constraints except for constraint 23. This is the constraint linking the processing of jobs and batches together. To overcome this problem, a repair function is introduced. This function ensures that an infeasible individual X is repaired such that it becomes (close to) feasible. So, the repair function will guide the algorithm towards the feasible region [92]. It does this in two ways. Firstly, matrix $\mathbf{D}$ containing all x_{jb} values is repaired. The repair function makes sure that each job is assigned to a batch, or each row in the matrix contains only one positive entry. Note that this repair does not necessarily make matrix $\mathbf{D}$ feasible. Secondly, matrix $\mathbf{T}$ containing all t^s_{jom} values is repaired. The repair function will first make sure that all operations not present in a jobs' routing are set to zero. Recall that a jobs routing contains all operations needed to produce the job. Then

it will repair starting times per operation. To do so, the function uses the production sequence of batches at the first operation within the jobshop, laser cutting, as a guideline. Starting time of the first batch in the sequence is set to 1. Starting time of the second batch is set to the sum of starting time, setup time and processing time of the first batch. This is repeated for all other batches within the sequence. For the remaining operations, batches/jobs are handled using the sequence determined at the first operation. Here, the starting time of an operation is the maximum between time at which a job finishes its last operation (job availability) and the time at which the machine finishes processing the previous job (machine availability). The repair function is depicted in algorithm 1. With the proposed repair function in place, the GDE3 algorithm is able to solve the dummy problem to optimality.

Algorithm 1: FUNCTION repair

```
Input: infeasible population
Output: feasible population
for individual in population do
    if individual is infeasible then
        Get D and T from individual
        Get set of jobs from D
        J ← set of jobs
        Get set of batches from D
        B ← set of batches
        for job in J do
            if job in multiple batches then
                Randomly delete job from batches until job in one batch only
            else if job not in batch then
                Assign job to a randomly chosen batch
        Get starting times of batches t^s_{b,o,m} from T
        laser_sequence ← sort(t^s_{b,o=1,m})
        sorted_batches ← sort(B) based on laser_sequence
        sorted_jobs ← sort(J) based on laser_sequence
        for batch in sorted_batches do
            Set t^s_{b,o=1,m} of first batch to 1
            Set t^s_{b,o=1,m} of remaining batches using production times
            Set t^s_{b,o=2,m} to max(t^s_{b,o=1,m}, t^s_{b-1,o=2,m})
        for o > 2 do
            for job in sorted_jobs do
                if o not in routing of job then
                    Set t^s_{j,o,m} to 0
                else if o in routing of job then
                    Use laser_sequence to set t^s_{j,o,m} to max(t^s_{j,o-1,m}, t^s_{j-1,o,m})
return population
```

Next to the repair function, we also introduce a termination criterion for the algorithm. For this research, the number of generations the algorithm can iterate is chosen as termination criterion. Running the algorithm too short may lead to unsatisfactory results, whereas running it too long will waste computational resources. For each problem instance, the correct number of generations needs to be determined. Incorporating a termination criterion based on algorithm performance is devoted to future research.

5.5 Decision space normalization

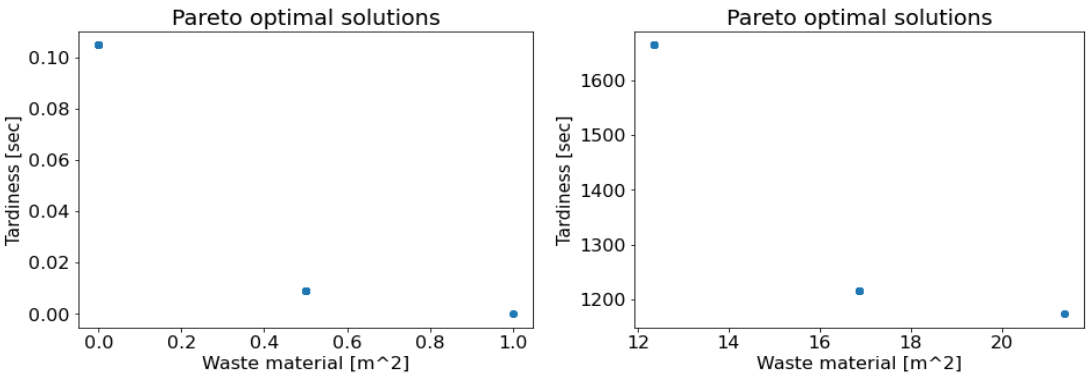
The JSSP considering batches has differently scaled objectives. Where the number of used batches in set B, which will be denoted as waste material, is measured in square meters with a approximated range of 0 to 100, job tardiness is measured in seconds with a approximated range of 0 to multiple thousands. In order for a multi-objective optimization evolutionary algorithm (MOEA) such as GDE3 to evaluate differently scaled objectives equally, objective space normalization should be applied. The authors in [101] explain that in objective space normalization, all objective values are scaled between 0 and 1. To do so, information on the range of the Pareto front is needed. For this, the ideal and nadir points, which are determined per objective, are used. In case of a minimization objective, the ideal point z^{lb} is a lower bound of the objective and the nadir point z^{ub} an upper bound. Since the ideal and nadir points of the JSSP considering batches are not known a priori, estimated ideal and nadir points will be used. To estimate the ideal points z_i^{lb}, GDE3 is solved for each objective separately using standard control parameter settings $CR = 0.5$, $F = 1$ and $N = 5D$.

Estimating a nadir point is done using common sense. In case jobs are not combined in batches but instead each job is assigned to a individual batch, waste material will reach an upper bound. For job tardiness to reach an upper bound, we use a simple heuristic. GDE3 is given an upper bound for t^s_{jom}, being the staring time of operation o of job j at machine m. In the heuristic, we assign this upper bound to the final operation of job $j \in J$ having the earliest due date. So, the job with the earliest due date will finish processing last. Then, the job with the second earliest due date will be assigned the upper bound minus the processing time of the last operation of the former job. This process is continued until all jobs are assigned a starting time for their last operation.

When the estimated ideal and nadir points are obtained, the objective space can be normalized. This is done for each individual within the population using equation 31. Here, $\tilde{f}_i(x)$ denotes the ith normalized objective function and n is the number of objective functions within the model.

$$\tilde{f}_i(x) = \frac{f_i(x) - z_i^{lb}}{z_i^{ub} - z_i^{lb}}, i \in \{1, 2, ..., n\} \tag{31}$$

In order to test correctness of the estimates for ideal and nadir points and whether normalization has effect on algorithm performance, 2 runs for a small problem instance containing 5 jobs are conducted. The first run without objective space normalization, the second with normalization. Results are depicted in Figure 3 and Figure 4.



(a) Results with normalization (b) Results without normalization

Figure 3: Pareto sets with and without normalization

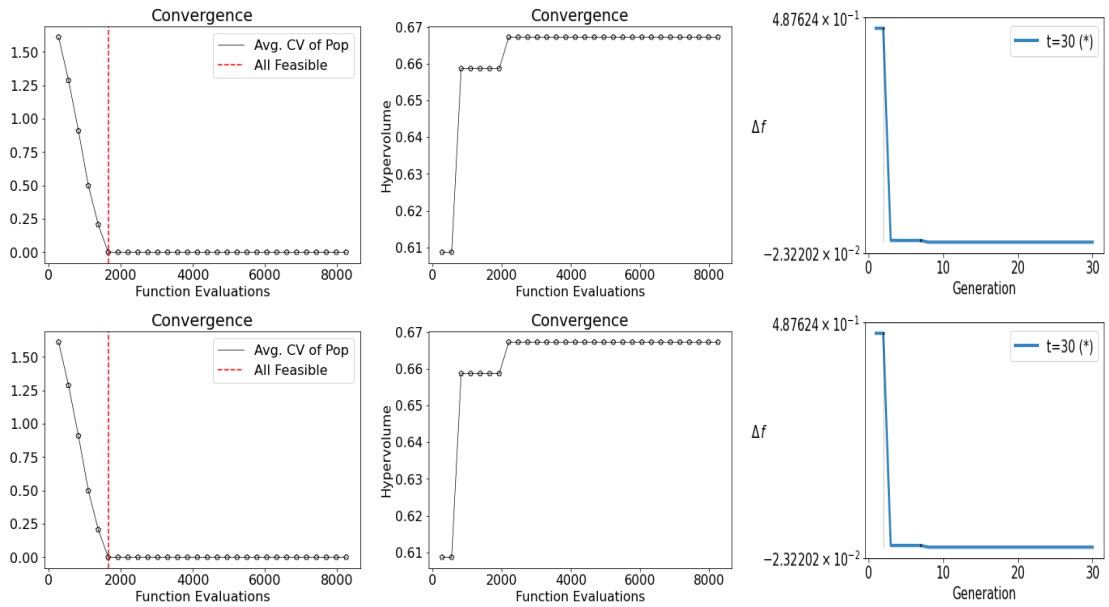


Figure 4: Performance indicators, above with normalization, below without normalization

Both the Pareto set plots and performance indicator plots show similar behaviour for both runs. Thus, normalization does not affect GDE3 when applied on the JSSP considering batches. Because estimates are used for decision space normalization, and thus a bias can be introduced, it is decided to continue the research without normalization of the decision space.

6 Numerical study: application to the Nooteboom jobshop

The proposed optimization approach is applied here to find a Pareto set containing solutions for the production schedule of the jobshop within Nooteboom Trailers BV. The jobshop within Nooteboom contains 6 machine sets performing 6 different operations. The different machine sets together with the corresponding operation index, type of processing and type of setup are listed in Table 1.

Table 1: Machine sets present in Nooteboom jobshop

Operation Index	Operation name	Process type	Setup type
1	Laser cutting	Batch	Sequence independent
2	Picking	Batch	Sequence independent
3	Milling	Job	None
4	Tapping	Job	None
5	Bending	Job	Sequence dependent
6	Logistics	Job	None

It is assumed that each machine set contains one machine. In reality, this assumption holds for all machine sets except for set 5, which in reality contains two machines. Furthermore, it is assumed that all machines are available at the start of the model, machines do not fail and worker availability is not considered. Machines performing batch processing incur sequence independent setup times. Operation 5, bending, incurs sequence dependent setup times, meaning that the setup depends on both the current and the preceding job processed by a machine within machine set 5. In subsection 6.1, we will discuss the data set containing jobs produced in the jobshop provided by Nooteboom. In subsection 6.2, the current policy for solving the JSSP considering batches is discussed. Furthermore, the implementation of this policy within a Discrete Event Simulation (DES) is discussed in this section. Results of the simulation will serve as benchmark for the performance of GDE3. After discussing the results of the parameter optimization in subsection 6.3, results of solving the JSSP considering batches using GDE3 are discussed in subsection 6.4. The performance of GDE3 is quantified using performance indicators discussed in subsection 5.3. In subsection 6.5 the performance of GDE3 is compared with two benchmarks, being the current policy used to solve the JSSP considering batches and the optimal solution of the integer linear program discussed in subsection 4.1. Lastly, in subsection 6.6, the effects of the safety factor regarding batch area on both batch feasibility and algorithm performance are analysed.

6.1 Data set

A data set containing 20305 jobs is available. This is the equivalent of 5 months of production, where production took place in 2 shifts of 8 hours per day. Per job, the data set contains the features release date, due date, production plan (operations needed to process the job), process time per production step (except for logistics), material property, thickness property and area. It should be noted that process times per production step are estimated times. For the logistics operations, data on process times is not available. However, data on the hours worked at the logistics operation during the period the jobs have been processed is available. The average time a job spends at the logistics operation is calculated by dividing the total hours worked at logistics during the 5 month period by the number of jobs served during the 5 month period. The estimated time is then added to the data set to ensure each production step has a process time assigned.

As mentioned before, the data set contains a release date per job. An analysis of release dates shows that multiple jobs are released simultaneously. On average, every 1.6 days a release

event occurs composed of on average 183 jobs. In addition, at each release event, an average of 79 jobs has not started processing at the laser cutter. This results in a total of 262 jobs to be scheduled per release event. At the start of each production shift, we want to optimize the production schedule. If a release event would occur every shift, this would result in an average of 80 jobs to be scheduled per shift. If we look at single release events, the data set shows that the smallest event only contains one job. Events with 10 jobs occur multiple times within the data set. In order to limit computational time, we will focus on problem instances containing 10 jobs in the remainder of this section.

Next to a release date per job, the data set also contains a due date per job. These due dates are based on a stochastic production environment. Because the situation analysed in this paper is deterministic, due dates have to be adjusted to reflect the situation. This is done per job, making use of the sum of all production and setup times of this job. The due date per job is set to this sum.

6.2 Current policy

In order to analyse the current production process within the Nooteboom jobshop, a deterministic DES was built. A deterministic simulation is used, because data on stochasticity (eg machine availability) is not available. The simulation is used to simulate the current policy used by the company to solve the JSSP considering batches. Thus, jobs need to be assigned to batches, batches need to be assigned to machines, jobs need to be assigned to machines and the processing sequence per machine needs to be determined.

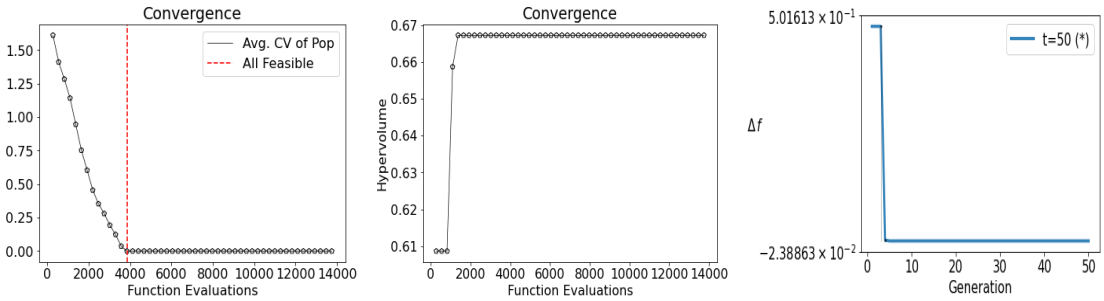
This is done using the following policy. Assignment of jobs to batches is done based on EDD. At each release event, the released jobs are combined with the jobs which have not started processing yet. These are jobs which are already batched but of which production of the first operation has not yet started. The job with the earliest due date amongst the combined jobs is assigned to a new batch. Consecutively, the batch is filled with available jobs. Similarly to the problem described in section 3, material and thickness properties of all jobs assigned to a batch should be equal. In addition, the total area of jobs assigned to a batch must be smaller than a certain fraction of the available area. After batch definition, both batch and job assigned and machine sequencing are conducted using an EDD heuristic. The policy is implemented in a DES using the event types job release, batching, server arrival, job arrival and job departure. The implemented policy is depicted in algorithm 2 in Appendix A.

The simulation was validated based on the definition of batches and the processing of jobs, using the parameters waste material and tardiness of jobs. If the fraction of usable batch area is set to the average waste percentage from current practise, the simulation results in a waste percentage deviating just 1 percent from current practise. Thus, regarding batch definition, the simulation is valid. Tardiness of jobs resulting from the simulation is lower than tardiness in current practise. This is caused by the stochastic nature of real life production and the deterministic nature of the simulation. Thus, the simulation behaves as expected regarding tardiness of jobs.

6.3 Results parameter optimization

In order to optimize the control parameters of the proposed GDE3 algorithm, a combination of the tuning methods multi-fidelity optimization and grid search is used. In multi-fidelity optimization, a subset of the data is used. Instead of instances of 10 jobs, which will be used to test GDE3, we will use instances of 5 jobs durting parameter optimization. The bounds of the control parameters are set according to the intervals proposed by[93]. So $N \in [275, 1650]$, $CR \in [0, 1]$ and $F \in (0, 2]$. To obtain the first grid, the parameter bounds are used in combination with a step size of 5. This results in a three dimensional grid of size 5^3. Before the grid search is executed, the termination criterion is determined using the performance measures convergence

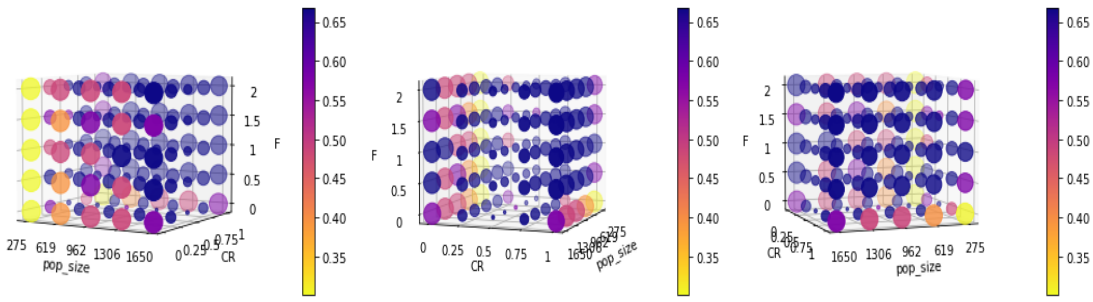
of population feasibility, hypervolume indicator and running metric. The results are depicted in Figure 5.



(a) Feasibility (b) Hypervolume indicator (c) Running metric

Figure 5: Grid search termination criterion

The plots in Figure 5 are created by running GDE3 for 50 generations. Control parameters are set to $N = 275$, $CR = 0.5$ and $F = 1$. Furthermore, the safety factor regarding batch area is set to one, meaning that it has no influence. Subplot (a) shows the convergence of population feasibility. After 3850 function evaluations, or 14 generations, all individuals within the population are feasible. Subplot (b) shows the convergence of the hypervolume indicator. After 1375 function evaluations, or 5 generations, the hypervolume indicator stopped converging and thus the algorithm reached an optimum. The running metric, depicted in subplot (c) shows the same result as subplot (b). With computational time of the grid search in mind, it is decided to set the termination criterion to 30 generations. It is expected that GDE3 has converged to an optimum before this point. In order to account for the stochastic nature of GDE3, 5 independent runs with different random seeds are performed. The results are averaged over the runs for each parameter combination. Results of the first grid search are visually presented in Figure 6.



(a) View 1 (b) View 2 (c) View 3

Figure 6: Grid search results $N \in [275, 1650]$, $CR \in [0, 1]$ and $F \in (0, 2]$

The subplots show different view angles of the results of the grid search. The control parameters are visible on the axes of the plots. The color of a dot refers to its hypervolume indicator score, the size refers to the speed at which it converged to feasibility. We want both performance measures to be as high as possible. In terms of the plot, this translates to large blue dots. We will analyze the three parameters consecutively, starting with crossover rate CR.

When analysing the different planes displaying CR, best seen in subplot (b), we notice a clear difference in dot sizes. Points in the plane $CR = 0$ and $CR = 1$, the extremes of this parameter, are largest. Planes in between the extremes show decreasing dot sizes, independent of the other parameter values. When analysing the planes $CR = 0$ and $CR = 1$, best seen in respectively subplots (a) and (c), with respect to dot color, we notice that the first plane contains mostly light colored dots, whereas the second plane contains mostly blue dots. The planes in between

$CR = 0$ and $CR = 1$ show mainly blue dots, so for these values of CR, algorithm performance with respect to hypervolume indicator is best. We can conclude that algorithm performance is best when CR lays between $CR = 0.8$ and $CR = 1$.

Mutation factor F is analysed next, starting with dot size. The different planes displaying F, best seen in subplot (b), show that all planes are similar with respect to dot size. This means that F does not affect convergence speed of the algorithm. When analysing dot color, differences in results are seen in the planes $CR = 0$ and $CR = 1$, best seen in respectively subplots (a) and (c). As $CR = 1$ yielded best algorithm performance with respect to CR, we will focus on this plane when analysing F. The plane is best seen in subplot (c). Results in this plane clearly show that a value near zero yields lower hypervolume indicator values. For values 0.5, 1.5 and 2, results are similar. For $F = 1$, algorithm performance is slightly higher for a low population size. Thus, for F, algorithm performance is best for the value 1.

The last parameter to analyse is population size N. Again, dot size is analysed first. Similarly to F, the different planes displaying N, best seen in subplot (b), show that all planes are similar with respect to dot size. Thus, population size does not affect convergence speed of the algorithm. Also similar to F, differences in dot color are seen in the planes $CR = 0$ and $CR = 1$, best seen in respectively subplots (a) and (c). Again, we will focus on the plane $CR = 1$, best seen in subplot (c). Here, we see that all values of N yield equal results except for $N = 250 = 5D$, which yield a lower result regarding hypervolume indicator values. This indicates that algorithm performance is best for $N > 5D$.

As algorithm performance only deteriorates slightly when population size decreases, we want to investigate to which extend this parameter can be decreased. Therefore, a second grid search is executed to investigate algorithm performance for population size values between $1D$ and $5D$. This translates to $N \in [55, 275]$. The other two parameters are set to $CR \in [0.8, 1]$ and $F \in (0.5, 1.5]$. Again, before the grid search is executed, the termination criterion is determined. Results of the second grid search are displayed and discussed in Appendix B. We can conclude that although good hypervolume indicator scores can be obtained when population size is decreased to $3D$, convergence speed will decrease. Therefore, it is decided to continue this research with a population size of $5D$, in combination with $CR = 1$ and $F = 1$.

6.4 Results optimization with GDE3

In this section, we will discuss results of solving the JSSP considering batches using the proposed algorithm GDE3. We will analyse algorithm performance based on performance measures discussed in subsection 5.3.

GDE3 in combination with the optimized parameters is used to solve three different problem instances of 10 jobs each. Three different instances are used to both give a good representation of instances occurring within the data set and to analyse algorithm robustness under different input conditions. In the first instance, variety amongst material and thickness properties of jobs is high. This is combined with a high number of operations in the production plans of the jobs. The second instance contains jobs with no variety in material and low variety in thickness properties. Furthermore, jobs have a low number of operations in their production plan. The third instance can be considered as a combination of the first and second instance. Jobs have low variety amongst material and thickness properties combined with a high number of operations in the production plans. Together, the three instances give a good representation of problem instances occurring within the data set and thus in current practise.

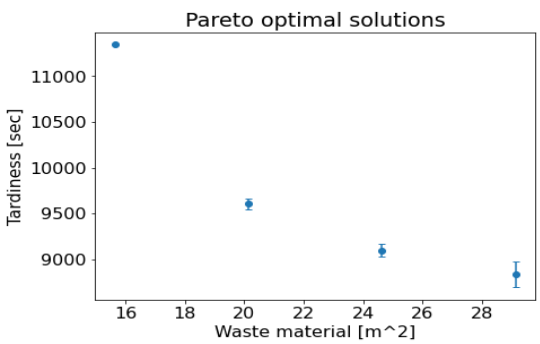
Prior to optimizing the different problem instances, the termination criterion per instance is determined. The plots resulting from running GDE3 for 100 generations with parameter settings $CR = 1$, $F = 1$ and $N = 800$ are shown in Appendix C. Results indicate that termination criterions of respectively 50, 60 and 60 generations are expected to yield results of good quality.

With the specified parameters, the three different problem instances of 10 jobs are solved.

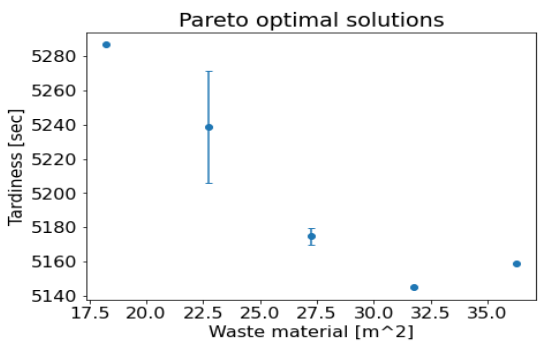
Because of the stochastic nature of GDE3, five runs are executed per problem instance. Five runs is sufficient as the variability of tardiness over the runs proved to be sufficiently low. The Pareto sets per problem instance are depicted in Figure 7. The Pareto sets display the mean and standard deviation for job tardiness per waste material level found by the algorithm. The Pareto set of the first problem instance shows increasing standard deviation of tardiness with an increasing level of waste material. This can be attributed to the number of jobs per batch. For low levels of waste material, the number of jobs per batch will be high. All jobs assigned to a batch will have equal finishing times for the first 2 operations. This leaves less room for variation in starting times for the remaining operations. For high levels of waste material, the number of jobs per batch will be low. This means that variety in starting times of all operations will be higher. Furthermore, this Pareto set shows a much larger scale for job tardiness compared to the other sets. This is attributed to the production times of jobs for the first operation being significantly larger compared to the other two problem instances.

The Pareto set of the second problem instance shows lower standard deviations compared to the other instances. Where standard deviations of the other two instances have a maximum value of around 200, standard deviation for this instance has a maximum of around 50. This is explained by the fact that this problem instance contains jobs with a low number of operations per job, resulting in less room for variation in starting times as operations are limited. Furthermore, this instance has the highest number of optimal solutions. Due to jobs with no variety in material and low variety in thickness properties within this instance, a lot of job combinations within batches are possible.

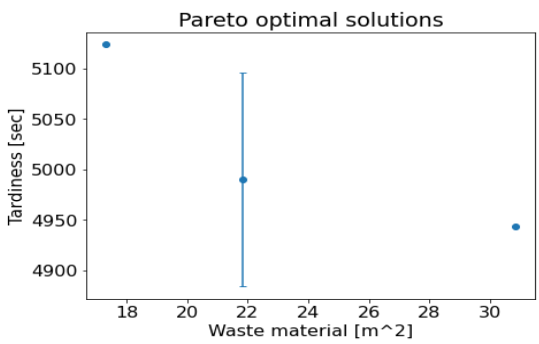
The Pareto set of the third problem instance shows a maximum value of around 200 for standard deviation, which is explained by the high number of operations per job within this instance. The Pareto set also shows that, in case low variety amongst material and thickness properties of jobs is combined with a high number of operations per job, the number of solutions within the Pareto set will be lower.



(a) Problem instance 1



(b) Problem instance 2



(c) Problem instance 3

Figure 7: Pareto set per problem instance

In order to determine the quality of the different Pareto sets, we will use the performance measures convergence speed with regards to population feasibility and hypervolume indicator. Running metric is excluded, because we run each problem instance 5 times in parallel. This makes it impossible to save the required data to obtain the running metric. For each run, plots of the 2 performance measures and the found Pareto set, including feasible solutions found, are displayed in Appendix C. The mean and standard deviation of the two performance measures per problem instance are given in Table 2. As problem complexity increases from instance one to instance three, hypervolume indicator values decrease. Although it becomes harder for the algorithm to find good quality solutions if problem complexity increases, a higher number of generations could solve this problem. Instance two used the most time to converge to feasibility. Recall that the repair function does not necessarily repair the assignment of jobs to batches to feasibility. As instance two contains jobs with no variety in material and low variety in thickness properties, a lot of job combinations within batches are possible. Therefore, it will cost the repair function more time to guide this instance to the feasible region compared to the other two instances.

Table 2: Performance indicator scores per problem instance

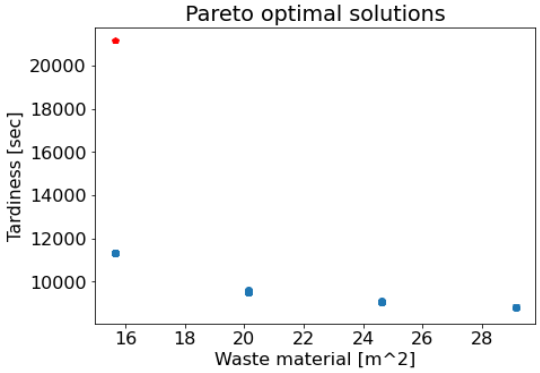
		Mean	St. dev.
Instance 1	Convergence speed	11	0.63
	HV indicator	0.74	0.03
Instance 2	Convergence speed	19.8	1.33
	HV indicator	0.53	0.23
Instance 3	Convergence speed	14	0.63
	HV indicator	0.30	0.19

6.5 Comparison of GDE3 with benchmarks

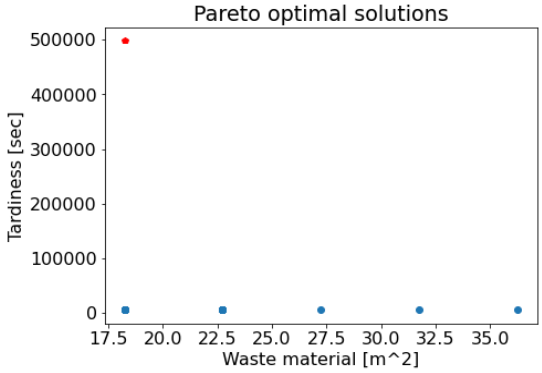
In this section, we will compare results of GDE3 with two benchmarks. The first benchmark is the current policy to solve the JSSP considering batches, discussed in subsection 6.2. The second benchmark is the optimal solution of the integer linear program discussed in subsection 4.1. The optimal solution is obtained using the solver Gurobi [102]. Note that for the comparison with Gurobi, problem instances are decreased to 5 jobs due to run time of the solver.

Per problem instance, the result from the current policy (red) is plotted together with the Pareto set found by GDE3 (blue). Results are displayed in Figure 8. It can be seen that GDE3 outperforms the current policy in terms of tardiness for all problem instances. This means that GDE3 is capable of scheduling the jobs such that the makespan of all jobs is minimized. In terms of waste material, the solution found by EDD heuristic aligns with the best solution found by GDE3. However, GDE3 is able to combine a low level of waste material with a significantly lower level of tardiness compared to current policy. Next to better performance in terms of tardiness, GDE3 provides a set of Pareto optimal solutions. This enables a decision making process in which the trade-off between objectives can be decided upon. Thus, the decision maker can align the schedule to fit the current wishes of the company regarding objectives.

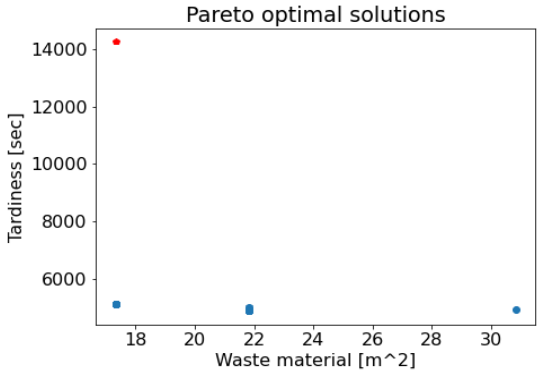
The different plots clearly show that the current policy solely focuses on minimizing waste material. When a batch is released to the jobshop, it is not known a priori what the effect of this batch on the second objective, job tardiness, will be. GDE3 does provide this insight to the company. Furthermore, GDE3 can support in determining required capacity needed within the jobshop.



(a) Problem instance 1



(b) Problem instance 2



(c) Problem instance 3

Figure 8: Pareto set and EDD heuristic solution per problem instance

To solve the integer linear program discussed in subsection 4.1, the solver Gurobi is used. When Gurobi needs to solve a MOO problem, it will solve the objectives consecutively. Gurobi is ran twice per problem instance in order to put equal emphasis on both objectives. As stated before, the problem instances are decreased before we solve them using Gurobi. The reason is run time of Gurobi. Figure 9 shows that run time increases exponentially when problem size increases. Solving a problem instance of 10 jobs is computationally too expensive, therefore we will focus on problem instances of 5 jobs. Results of GDE3 and Gurobi are displayed in Figure 10.

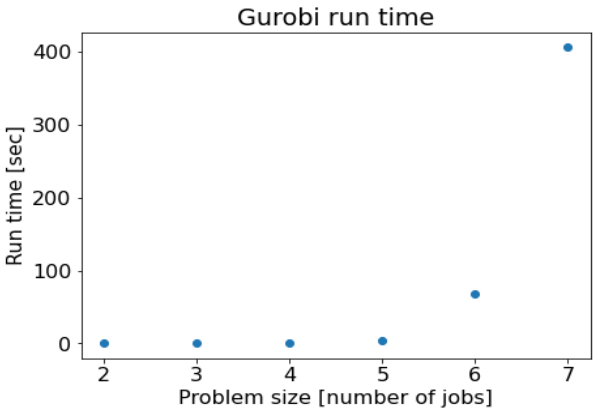
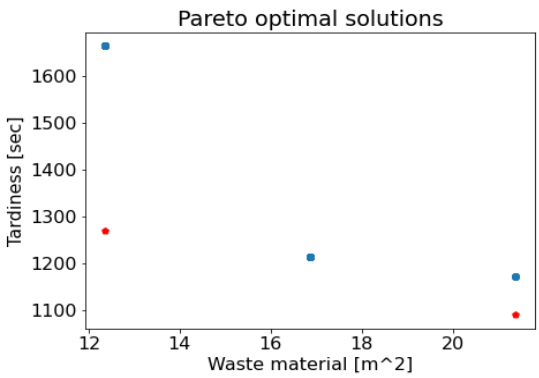
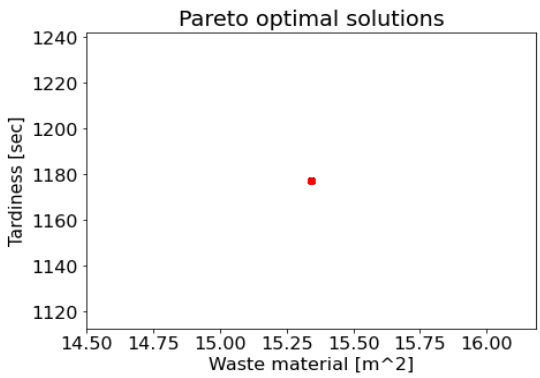


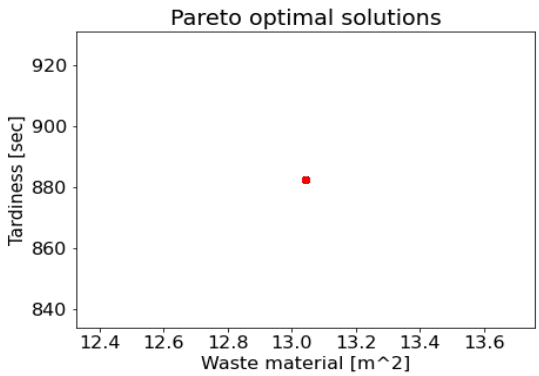
Figure 9: Gurobi run time



(a) Problem instance 1



(b) Problem instance 2



(c) Problem instance 3

Figure 10: Pareto set and Gurobi solver solution per problem instance

Results show that Gurobi finds a slightly better solution than GDE3. This can be attributed to the repair function. When starting times of batches and jobs are repaired, the sequence at the first operation, which the algorithm determined, is used to repair the rest of the times. This sequence may not be the optimal sequence, as can be seen in subplot (a). In case there exists only one feasible sequence, GDE3 finds the same solution as Gurobi. This is the case in subplots (b) and (c).

We can conclude that GDE3 outperforms the current policy and performs only slightly worse than Gurobi solver in terms of algorithm performance. Although computational time of GDE3 is better compared to Gurobi solver, it is not fast enough to solve larger problem instances. For the problem instances of 10 jobs, run time is between 35 and 45 minutes, which is acceptable. However, for larger instances, run time of GDE3 needs to be improved. The main reason why GDE3 becomes slow for larger problem instances is the definition of an individual within the population. The dimensions of an individual equal $D = n * o + n^2$, where n equals the number of jobs and o the number of operations. The term n^2 causes dimensions of an individual to increase quadratic. If this is altered for larger problem instances, it is expected that GDE3 is capable of finding good solutions in an acceptable time frame.

6.6 Results safety factor batch area

Up until now, the safety factor regarding batch area has not been used. Recall that the safety factor is introduced to compensate for omitting placement of parts and cutting seems. Thus, the safety factor should prevent jobs which could form a feasible batch based on area but would form an infeasible batch based on geometry from being batched. To test the effects of the safety factor, a new problem instance is introduced. This instance has the feature that the included jobs could be batched based on job area, but not on job geometry. This situation is

visually represented in Figure 11.

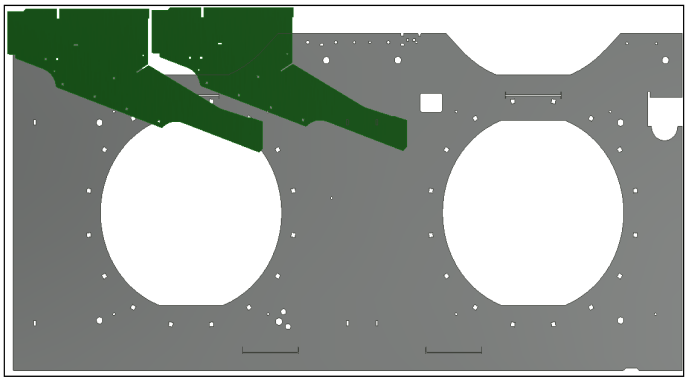
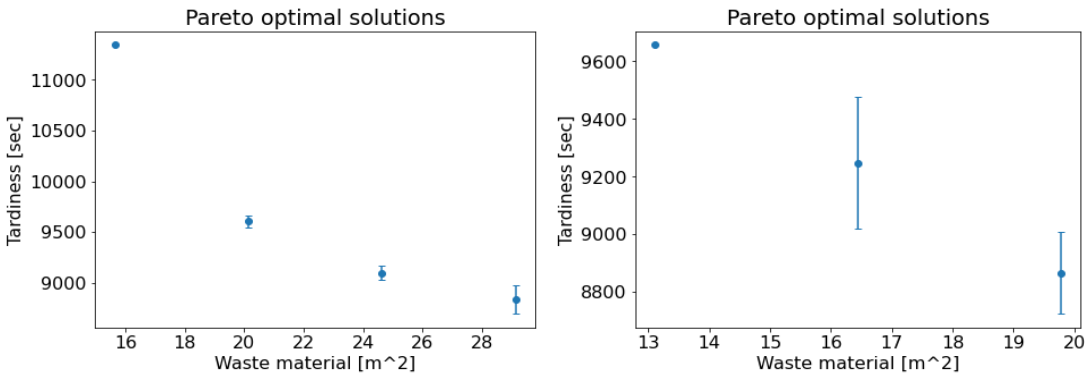


Figure 11: Geometric infeasible batch

When we run GDE3 with a safety factor of one, as such it has no influence, the algorithm will assign the two jobs to the same batch. When the safety factor is set to 0.74, which translates to limiting the algorithm to using a maximum of 74 percent of the usable batch area, the jobs are assigned to different batches. So, in case of this test instance the safety factor gives the desired result. To see the impact of the safety factor on algorithm performance on a different problem instance, problem instance 1 is optimized again. Results are displayed in Figure 12.



(a) Without safety factor

(b) With safety factor

Figure 12: Pareto set with and without safety factor

Note that the values of waste material can not be compared between the plots because of the safety factor. We see that with the safety factor in place, GDE3 finds one solution less. If we analyse the solution which is excluded due to the safety factor, we see that this would have been a feasible batch. So the safety factor is preventing an infeasible batch to be formed by the algorithm but at the same time it prevents a feasible batch to be formed as well. Although the safety factor is a simple way to prevent infeasible batches, it also limits the algorithm. Therefore, it is recommended to use the algorithm without the safety factor.

7 Managerial insights

The proposed Generalized Differential Evolution 3 (GDE3) algorithm allows solving the jobshop scheduling problem considering batches and provides a set of Pareto optimal solutions to the decision maker. Using the set of solutions, the decision maker can decide on the trade-off between objectives. Thus, the decision maker can align the production schedule to fit the current wishes of the company regarding objectives.

Under the assumptions made within this research, GDE3 outperforms the current policy in terms of tardiness for all problem instances. This means that GDE3 is capable of scheduling the jobs such that the makespan of all jobs is lower compared to current practice. Thus, with the same amount of resources, GDE3 could either finish the same set of jobs faster or schedule a higher amount of jobs within the same time frame, taking capacity restrictions into account. In terms of waste material, the solution found by EDD heuristic aligns with the best solution found by GDE3. However, GDE3 is able to combine a low level of waste material with a significantly lower level of tardiness compared to current policy. Although performance of GDE3 is slightly lower compared to the optimal solution provided by the Gurobi solver, computational time of GDE3 is much lower. So, GDE3 is capable of finding near optimal solutions in an acceptable time frame.

The Pareto front, next to enabling a decision making process, also facilitates insights in the production process before the start of the process. This enables better estimation of required capacity within the jobshop. Where the current policy is to react to capacity shortage, GDE3 enables taking preventive measures. The algorithm also provides insights in material usage. These insights can be used to prevent material shortages and maintain a lower safety stock. Lastly, the algorithm provides insights in scheduled machine utilization. In current practise, we see high variation in utilization rates. With a proper schedule, rates can be balanced over time resulting in a more stable production process.

8 Conclusions

In this study, we applied the metaheuristic Generalized Differential Evolution 3 (GDE3) to solve the jobshop scheduling problem considering batches. GDE3 is applied to several representative real-world problem instances. We can draw three main conclusions. Firstly, GDE3 outperforms current practise in terms of job tardiness. In terms of waste material, GDE3 finds the same solution as in current practise, but in combination with much lower job tardiness. Although performance of GDE3 is slightly lower compared to the Gurobi solver, computational time of GDE3 is much lower. So, GDE3 is capable of finding near optimal solutions in an acceptable time frame. Secondly, GDE3 provides a set of Pareto optimal solutions. This set provides insights in how different production schedules affect production objectives and enables to substantiate a decision for a certain trade-off between objectives. Furthermore, the set provides insights in the production process before the start of production. Thirdly, compared to relevant literature, this research accounts for setup times, which in a jobshop environment with a high product mix is a very important factor in schedule computation.

Although the problem instances used in this research are representative for real world instances, they are amongst smaller instances. In future research, it would be of interest to alter the algorithm such that it will need less run time and thus can solve larger problem instances. Parameter control and proper termination criterion could also add to limiting computational time of the algorithm, so these are also interesting fields of future research.

It would also be of interest to investigate how the assumption of a one dimensional bin packing problem can be incorporated in the model such that batch feasibility is taken into account. In this research, we propose to use a safety factor on batch area for this purpose. Although enhancing batch feasibility, the factor also limits the algorithm in forming batches and thus is not an ideal solution.

Furthermore, it would be interesting to see the effects of adding additional objectives to the model. For instance, we could add the objective of minimizing earliness of jobs to see the effect on work in progress. In current times of climate change and high energy prices, it could be interesting to add the objective of minimizing power consumption.

References

[1] S. S. Kamble, A. Gunasekaran, and S. A. Gawankar, "Sustainable industry 4.0 framework: A systematic literature review identifying the current trends and future perspectives," *Process safety and environmental protection*, vol. 117, pp. 408–425, 2018.

[2] M. C. Duman and B. Akdemir, "A study to determine the effects of industry 4.0 technology components on organizational performance," *Technological Forecasting and Social Change*, vol. 167, p. 120 615, 2021.

[3] S. S. Kamble, A. Gunasekaran, A. Ghadge, and R. Raut, "A performance measurement system for industry 4.0 enabled smart manufacturing system in smmes-a review and empirical investigation," *International journal of production economics*, vol. 229, p. 107 853, 2020.

[4] Q. Zhang and M. Tseng, "Modelling and integration of customer flexibility in the order commitment process for high mix low volume production," *International Journal of Production Research*, vol. 47, no. 22, pp. 6397–6416, 2009.

[5] S. A. Irani, *Job Shop Lean: An Industrial Engineering Approach to Implementing Lean in High-mix Low-volume Production Systems*. CRC Press, 2020.

[6] R. A. Liaqait, S. Hamid, S. S. Warsi, and A. Khalid, "A critical analysis of job shop scheduling in context of industry 4.0," *Sustainability*, vol. 13, no. 14, p. 7684, 2021.

[7] H. Xiong, S. Shi, D. Ren, and J. Hu, "A survey of job shop scheduling problem: The types and models," *Computers & Operations Research*, p. 105 731, 2022.

[8] A. M. Ham and E. Cakici, "Flexible job shop scheduling problem with parallel batch processing machines: Mip and cp approaches," *Computers & Industrial Engineering*, vol. 102, pp. 160–165, 2016.

[9] J. W. Fowler and L. Mönch, "A survey of scheduling with parallel batch (p-batch) processing," *European Journal of Operational Research*, 2021.

[10] J. Pei, B. Cheng, X. Liu, P. M. Pardalos, and M. Kong, "Single-machine and parallel-machine serial-batching scheduling problems with position-based learning effect and linear setup time," *Annals of Operations Research*, vol. 272, no. 1, pp. 217–241, 2019.

[11] C. Gahm, A. Uzunoglu, S. Wahl, C. Ganschinietz, and A. Tuma, "Applying machine learning for the anticipation of complex nesting solutions in hierarchical production planning," *European Journal of Operational Research*, vol. 296, no. 3, pp. 819–836, 2022.

[12] B. Verlinden, D. Cattrysse, and D. Van Oudheusden, "Integrated sheet-metal production planning for laser cutting and bending," *International journal of production research*, vol. 45, no. 2, pp. 369–383, 2007.

[13] A. Rinciog, C. Mieth, P. M. Scheikl, and A. Meyer, "Sheet-metal production scheduling using alphago zero," in *Proceedings of the Conference on Production Systems and Logistics: CPSL 2020*, Hannover: Institutionelles Repositorium der Leibniz Universität Hannover, 2020.

[14] N. M. Cid-Garcia and Y. A. Rios-Solis, "Positions and covering: A two-stage methodology to obtain optimal solutions for the 2d-bin packing problem," *Plos one*, vol. 15, no. 4, e0229358, 2020.

[15] B. Verlinden, D. Cattrysse, H. Crauwels, and D. Van Oudheusden, "The development and application of an integrated production planning methodology for sheet metal working smes," *Production Planning and Control*, vol. 20, no. 7, pp. 649–663, 2009.

[16] H. Zhu, M. Chen, Z. Zhang, and D. Tang, "An adaptive real-time scheduling method for flexible job shop scheduling problem with combined processing constraint," *IEEE Access*, vol. 7, pp. 125 113–125 121, 2019.

[17] P. Helo, D. Phuong, and Y. Hao, "Cloud manufacturing–scheduling as a service for sheet metal manufacturing," *Computers & Operations Research*, vol. 110, pp. 208–219, 2019.

[18] S. Mirjalili and J. S. Dong, *Multi-objective optimization using artificial intelligence techniques*. Springer, 2020.

[19] H. Crauwels, B. Verlinden, D. Cattrysse, and D. Van Oudheusden, "Sheet-metal shop scheduling considering makespan and flow time criteria," *The Open Operational Research Journal*, vol. 4, no. 1, 2010.

[20] T. Sakaguchi, T. Murakami, S. Fujita, and Y. Shimizu, "A scheduling method with considering nesting for sheet metal processing," in *International Symposium on Flexible Automation*, American Society of Mechanical Engineers, vol. 45110, 2012, pp. 317–320.

[21] T. Sakaguchi, H. Ohtani, and Y. Shimizu, "A heuristic approach for integrated nesting and scheduling in sheet metal processing," in *IFIP International Conference on Advances in Production Management Systems*, Springer, 2015, pp. 226–234.

[22] T. Sakaguchi, T. Tanaka, Y. Shimizu, and N. Uchiyama, "A scheduling method using genetic algorithm and dispatching rule for sheet metal processing," in *2016 International Symposium on Flexible Automation (ISFA)*, IEEE, 2016, pp. 198–201.

[23] M. Caramia and P. Dell'Olmo, "Multi-objective optimization," in *Multi-objective management in freight logistics*, Springer, 2020, pp. 21–51.

[24] A. Schrijver, "On the history of combinatorial optimization (till 1960)," *Handbooks in operations research and management science*, vol. 12, pp. 1–68, 2005.

[25] V. Kenny, M. Nathal, and S. Saldana, "Heuristic algorithms," *Visited on Oct*, vol. 20, p. 2018, 2014.

[26] V. Maniezzo, T. Stützle, and S. Voß, *Matheuristics*. Springer, 2021.

[27] I. Boussaïd, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Information sciences*, vol. 237, pp. 82–117, 2013.

[28] S. Bandyopadhyay, S. Saha, U. Maulik, and K. Deb, "A simulated annealing-based multiobjective optimization algorithm: Amosa," *IEEE transactions on evolutionary computation*, vol. 12, no. 3, pp. 269–283, 2008.

[29] D. M. Jaeggi, G. T. Parks, T. Kipouros, and P. J. Clarkson, "The development of a multi-objective tabu search algorithm for continuous optimisation problems," *European Journal of Operational Research*, vol. 185, no. 3, pp. 1192–1212, 2008.

[30] A. P. Reynolds and B. De la Iglesia, "A multi-objective grasp for partial classification," *Soft Computing*, vol. 13, no. 3, pp. 227–243, 2009.

[31] A. Duarte, J. J. Pantrigo, E. G. Pardo, and N. Mladenovic, "Multi-objective variable neighborhood search: An application to combinatorial optimization problems," *Journal of Global Optimization*, vol. 63, no. 3, pp. 515–536, 2015.

[32] A. Jaszkiewicz, "Genetic local search for multi-objective combinatorial optimization," *European journal of operational research*, vol. 137, no. 1, pp. 50–71, 2002.

[33] M. J. Geiger, "Foundations of the pareto iterated local search metaheuristic," *arXiv preprint arXiv:0809.0406*, 2008.

[34] S. Iredi, D. Merkle, and M. Middendorf, "Bi-criterion optimization with multi colony ant algorithms," in *International conference on evolutionary multi-criterion optimization*, Springer, 2001, pp. 359–372.

[35] K. Doerner, W. J. Gutjahr, R. F. Hartl, C. Strauss, and C. Stummer, "Pareto ant colony optimization: A metaheuristic approach to multiobjective portfolio selection," *Annals of operations research*, vol. 131, no. 1, pp. 79–99, 2004.

[36] C. C. Coello and M. S. Lechuga, "Mopso: A proposal for multiple objective particle swarm optimization," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, IEEE, vol. 2, 2002, pp. 1051–1056.

[37] H. Han, W. Lu, and J. Qiao, "An adaptive multiobjective particle swarm optimization based on multiple adaptive methods," *IEEE transactions on cybernetics*, vol. 47, no. 9, pp. 2754–2767, 2017.

[38] M. R. Sierra and C. A. Coello Coello, "Improving pso-based multi-objective optimization using crowding, mutation and -dominance," in *International conference on evolutionary multi-criterion optimization*, Springer, 2005, pp. 505–519.

[39] B. Niu, H. Wang, J. Wang, and L. Tan, "Multi-objective bacterial foraging optimization," *Neurocomputing*, vol. 116, pp. 336–345, 2013.

[40] R. Hedayatzadeh, B. Hasanizadeh, R. Akbari, and K. Ziarati, "A multi-objective artificial bee colony for optimizing multi-objective problems," in *2010 3rd international conference on advanced computer theory and engineering (ICACTE)*, IEEE, vol. 5, 2010, pp. V5–277.

[41] R. Akbari, R. Hedayatzadeh, K. Ziarati, and B. Hassanizadeh, "A multi-objective artificial bee colony algorithm," *Swarm and Evolutionary Computation*, vol. 2, pp. 39–52, 2012.

[42] Y. Xiang, Y. Zhou, and H. Liu, "An elitism based multi-objective artificial bee colony algorithm," *European Journal of Operational Research*, vol. 245, no. 1, pp. 168–193, 2015.

[43] F. Freschi and M. Repetto, "Vis: An artificial immune network for multi-objective optimization," *Engineering optimization*, vol. 38, no. 8, pp. 975–996, 2006.

[44] P. A. Castro and F. J. V. Zuben, "Mobais: A bayesian artificial immune system for multi-objective optimization," in *International Conference on Artificial Immune Systems*, Springer, 2008, pp. 48–59.

[45] K. C. Tan, C. K. Goh, A. Mamun, and E. Ei, "An evolutionary artificial immune system for multi-objective optimization," *European Journal of Operational Research*, vol. 187, no. 2, pp. 371–392, 2008.

[46] W. Guo, L. Wang, and Q. Wu, "Numerical comparisons of migration models for multi-objective biogeography-based optimization," *Information Sciences*, vol. 328, pp. 302–320, 2016.

[47] E. Zitzler, M. Laumanns, and L. Thiele, "Spea2: Improving the strength pareto evolutionary algorithm," *TIK-report*, vol. 103, 2001.

[48] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[49] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints," *IEEE transactions on evolutionary computation*, vol. 18, no. 4, pp. 577–601, 2013.

[50] J. D. Knowles and D. W. Corne, "M-paes: A memetic algorithm for multiobjective optimization," in *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No. 00TH8512)*, IEEE, vol. 1, 2000, pp. 325–332.

[51] S. Huband, P. Hingston, L. While, and L. Barone, "An evolution strategy with probabilistic mutation for multi-objective optimisation," in *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, IEEE, vol. 4, 2003, pp. 2284–2291.

[52] C. Igel, N. Hansen, and S. Roth, "Covariance matrix adaptation for multi-objective optimization," *Evolutionary computation*, vol. 15, no. 1, pp. 1–28, 2007.

[53] P. Venkatesh and K. Y. Lee, "Multi-objective evolutionary programming for economic emission dispatch problem," in *2008 IEEE Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century*, IEEE, 2008, pp. 1–8.

[54] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm and evolutionary computation*, vol. 1, no. 1, pp. 32–49, 2011.

[55] F. Xue, A. C. Sanderson, and R. J. Graves, "Pareto-based multi-objective differential evolution," in *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, IEEE, vol. 2, 2003, pp. 862–869.

[56] J.-F. Qiao, Y. Hou, and H.-G. Han, "Optimal control for wastewater treatment process based on an adaptive multi-objective differential evolution algorithm," *Neural Computing and Applications*, vol. 31, no. 7, pp. 2537–2550, 2019.

[57] J. Zhang and A. C. Sanderson, "Jade: Adaptive differential evolution with optional external archive," *IEEE Transactions on evolutionary computation*, vol. 13, no. 5, pp. 945–958, 2009.

[58] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," *IEEE transactions on evolutionary computation*, vol. 10, no. 6, pp. 646–657, 2006.

[59] R. Tanabe and A. Fukunaga, "Success-history based parameter adaptation for differential evolution," in *2013 IEEE congress on evolutionary computation*, IEEE, 2013, pp. 71–78.

[60] R. Tanabe and A. S. Fukunaga, "Improving the search performance of shade using linear population size reduction," in *2014 IEEE congress on evolutionary computation (CEC)*, IEEE, 2014, pp. 1658–1665.

[61] A. P. Piotrowski, "L-shade optimization algorithms with population-wide inertia," *Information Sciences*, vol. 468, pp. 117–141, 2018.

[62] F. Lezama, J. Soares, R. Faia, T. Pinto, and Z. Vale, "A new hybrid-adaptive differential evolution for a smart grid application under uncertainty," in *2018 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2018, pp. 1–8.

[63] P. Civicioglu, E. Besdok, M. A. Gunen, and U. H. Atasever, "Weighted differential evolution algorithm for numerical function optimization: A comparative study with cuckoo search, artificial bee colony, adaptive differential evolution, and backtracking search optimization algorithms," *Neural Computing and Applications*, vol. 32, no. 8, pp. 3923–3937, 2020.

[64] K. M. Sallam, S. M. Elsayed, R. K. Chakrabortty, and M. J. Ryan, "Improved multi-operator differential evolution algorithm for solving unconstrained problems," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2020, pp. 1–8.

[65] S. Kukkonen and J. Lampinen, "Gde3: The third evolution step of generalized differential evolution," in *2005 IEEE congress on evolutionary computation*, IEEE, vol. 1, 2005, pp. 443–450.

[66] J. Vesterstrom and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," in *Proceedings of the 2004 congress on evolutionary computation (IEEE Cat. No. 04TH8753)*, IEEE, vol. 2, 2004, pp. 1980–1987.

[67] D. Zhang and B. Wei, "Comparison between differential evolution and particle swarm optimization algorithms," in *2014 IEEE International Conference on Mechatronics and Automation*, IEEE, 2014, pp. 239–244.

[68] Y. Wang, H. Liu, H. Long, Z. Zhang, and S. Yang, "Differential evolution with a new encoding mechanism for optimizing wind farm layout," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 3, pp. 1040–1054, 2017.

[69] M. Wagner, J. Day, and F. Neumann, "A fast and effective local search algorithm for optimizing the placement of wind turbines," *Renewable energy*, vol. 51, pp. 64–70, 2013.

[70] M. Wagner, K. Veeramachaneni, F. Neumann, and U.-M. O'Reilly, "Optimizing the layout of 1000 wind turbines," *European wind energy association annual event*, vol. 205209, 2011.

[71] H. Long, Z. Zhang, Z. Song, and A. Kusiak, "Formulation and analysis of grid and coordinate models for planning wind farm layouts," *IEEE access*, vol. 5, pp. 1810–1819, 2017.

[72] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *IEEE transactions on evolutionary computation*, vol. 10, no. 3, pp. 281–295, 2006.

[73] J. Sun, B. Feng, and W. Xu, "Particle swarm optimization with particles having quantum behavior," in *Proceedings of the 2004 congress on evolutionary computation (IEEE Cat. No. 04TH8753)*, IEEE, vol. 1, 2004, pp. 325–331.

[74] T.-q. Wu, M. Yao, and J.-h. Yang, "Dolphin swarm algorithm," *Frontiers of Information Technology & Electronic Engineering*, vol. 17, no. 8, pp. 717–729, 2016.

[75] F. Lezama, L. E. Sucar, E. M. de Cote, J. Soares, and Z. Vale, "Differential evolution strategies for large-scale energy resource management in smart grids," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2017, pp. 1279–1286.

[76] X.-S. Yang and S. Deb, "Cuckoo search via lévy flights," in *2009 World congress on nature & biologically inspired computing (NaBIC)*, Ieee, 2009, pp. 210–214.

[77] D. Karaboga and B. Akay, "A comparative study of artificial bee colony algorithm," *Applied mathematics and computation*, vol. 214, no. 1, pp. 108–132, 2009.

[78] P. Civicioglu, "Backtracking search optimization algorithm for numerical optimization problems," *Applied Mathematics and computation*, vol. 219, no. 15, pp. 8121–8144, 2013.

[79] A. Kumar, R. K. Misra, and D. Singh, "Improving the local search capability of effective butterfly optimizer using covariance matrix adapted retreat phase," in *2017 IEEE congress on evolutionary computation (CEC)*, IEEE, 2017, pp. 1835–1842.

[80] G. Zhang and Y. Shi, "Hybrid sampling evolution strategy for solving single objective bound constrained problems," in *2018 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2018, pp. 1–7.

[81] N. H. Awad, M. Z. Ali, and P. N. Suganthan, "Ensemble sinusoidal differential covariance matrix adaptation with euclidean neighborhood for solving cec2017 benchmark problems," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2017, pp. 372–379.

[82] A. W. Mohamed, A. A. Hadi, A. M. Fattouh, and K. M. Jambi, "Lshade with semi-parameter adaptation hybrid with cma-es for solving cec 2017 benchmark problems," in *2017 IEEE Congress on evolutionary computation (CEC)*, IEEE, 2017, pp. 145–152.

[83] D. Wilson, S. Rodrigues, C. Segura, *et al.*, "Evolutionary computation for wind farm layout optimization," *Renewable energy*, vol. 126, pp. 681–691, 2018.

[84] I. Loshchilov and F. Hutter, "Cma-es for hyperparameter optimization of deep neural networks," *arXiv preprint arXiv:1604.07269*, 2016.

[85] A. Kheiri and E. Keedwell, "A sequence-based selection hyper-heuristic utilising a hidden markov model," in *Proceedings of the 2015 annual conference on genetic and evolutionary computation*, 2015, pp. 417–424.

[86] C. Gahm, S. Wahl, and A. Tuma, "Scheduling parallel serial-batch processing machines with incompatible job families, sequence-dependent setup times and arbitrary sizes," *International Journal of Production Research*, pp. 1–24, 2021.

[87] S. Kim and P. Bobrowski, "Impact of sequence-dependent setup time on job shop scheduling performance," *The International Journal of Production Research*, vol. 32, no. 7, pp. 1503–1520, 1994.

[88] K. Deb, A. Sinha, and S. Kukkonen, "Multi-objective test problems, linkages, and evolutionary methodologies," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 2006, pp. 1141–1148.

[89] Q. Zhang, A. Zhou, and Y. Jin, "Rm-meda: A regularity model-based multiobjective estimation of distribution algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 1, pp. 41–63, 2008.

[90] A. Adebiyi and C. Ayo, "Portfolio selection problem using generalized differential evolution 3," *Applied Mathematical Sciences*, vol. 9, no. 42, 2015.

[91] B. Leite, A. O. S. da Costa, and E. F. da Costa Junior, "Multi-objective optimization of adiabatic styrene reactors using generalized differential evolution 3 (gde3)," *Chemical Engineering Science*, p. 118 196, 2022.

[92] J. Blank and K. Deb, "Pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020.

[93] S. Kukkonen and J. Lampinen, "An empirical study of control parameters for the third version of generalized differential evolution (gde3)," in *2006 IEEE International Conference on Evolutionary Computation*, IEEE, 2006, pp. 2002–2009.

[94] M. Cervenka and H. Boudna, "Visual guide of f and cr parameters influence on differential evolution solution quality," pp. 141–144, 2018.

[95] A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith, "Parameter control in evolutionary algorithms," in *Parameter setting in evolutionary algorithms*, Springer, 2007, pp. 19–46.

[96] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[97] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization.," *Journal of machine learning research*, vol. 13, no. 2, 2012.

[98] A. E. Eiben and S. K. Smit, "Parameter tuning for configuring and analyzing evolutionary algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 19–31, 2011.

[99] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms—a comparative case study," in *International conference on parallel problem solving from nature*, Springer, 1998, pp. 292–301.

[100] J. Blank and K. Deb, "A running performance metric and termination criterion for evaluating evolutionary multi-and many-objective optimization algorithms," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2020, pp. 1–8.

[101] L. He, H. Ishibuchi, A. Trivedi, H. Wang, Y. Nan, and D. Srinivasan, "A survey of normalization methods in multiobjective evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 6, pp. 1028–1048, 2021.

[102] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2023. [Online]. Available: `https://www.gurobi.com`.

A Appendix

A Discrete event simulation

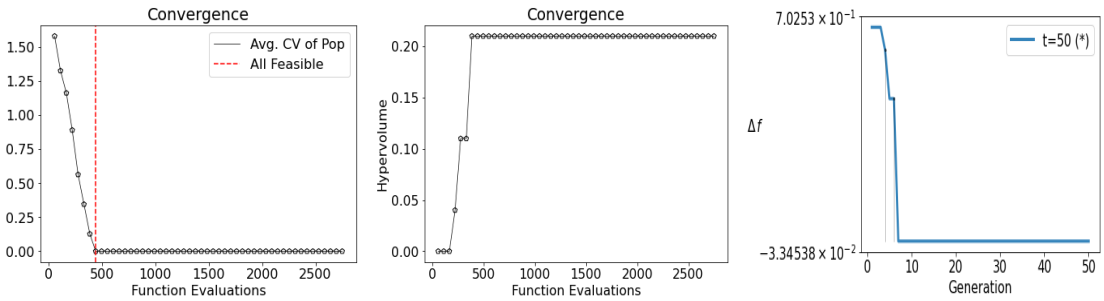
The algorithm below shows the current policy within the jobshop implemented within a discrete event simulation.

Algorithm 2: FUNCTION JSSP considering batches simulation

```
Input: Jobs
Output: Simulation results
Schedule release event for input jobs
while Not all orders produced do
    Get next event from FES
    if Event type is Release then
        Get jobs from event
        Get jobs which have not started production from laser queue
        Schedule Batching event for jobs and add it to FES
    else if Event type is Batching then
        Get jobs from event
        Create batches
        Register waste material
        Add batches to laser queue
    else if Event type is Server Arrival then
        Get current queue from event
        while queue is not empty do
            Get batch/job with EDD from queue
            Schedule departure event for batch/job and add it to FES
        if queue is empty then
            Schedule server arrival event and add it to FES
    else if Event type is Arrival then
        Get batch/job from event
        Get current queue from event
        Add batch/job to current queue
    else if Event type is Departure then
        Get batch/job from event
        Get current queue from event
        if Job finished processing then
            Remove job from current queue
            Register tardiness of job
        else if Batch/job needs further processing then
            Remove job from current queue
            Schedule arrival event of job in next queue and add it to FES
return (Total waste material, total tardiness)
```

B Parameter optimization

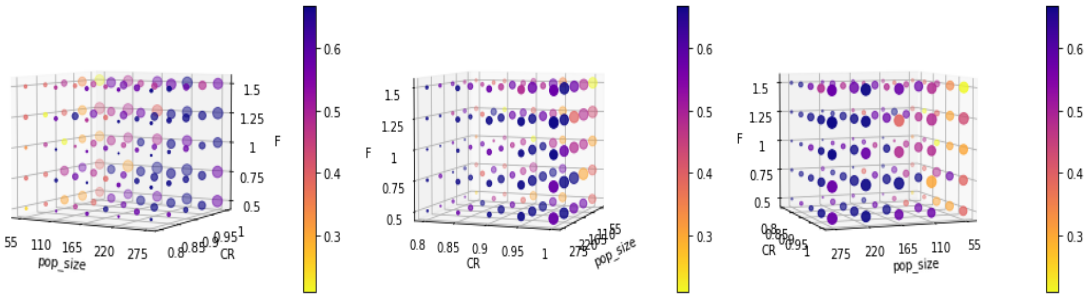
A second grid search is executed to investigate algorithm performance for population size values between $1D$ and $5D$. This translates to $N \in [55, 275]$. The other two parameters are set to $CR \in [0.8, 1]$ and $F \in (0.5, 1.5]$. Before the grid search is executed, the termination criterion is determined by running GDE3 for 50 generations. Results are depicted in Figure 13.



(a) Feasibility (b) Hypervolume indicator (c) Running metric

Figure 13: Grid search termination criterion

Subplot (a) shows that after 440 function evaluations, or 8 generations, all individuals within the population are feasible. Subplot (b) shows that the hypervolume indicator stopped converging after 385 function evaluations, or 7 generations. The running metric, depicted in subplot (c) shows the same result as subplot (b). We therefore set the termination criterion to 20 generations. The results of the second grid search are depicted in Figure 14.



(a) View 1 (b) View 2 (c) View 3

Figure 14: Grid search results $N \in [55, 275]$, $CR \in [0.8, 1]$ and $F \in (0.5, 1.5]$

Similarly to the first grid search, the color of a dot refers to its hypervolume indicator score and the size to the speed at which it converged to feasibility. Although the scale of the color of both searches is equal, the scale of dot size differs between the searches. Again, the three control parameters will be analysed sequentially in the order CR, F and N.

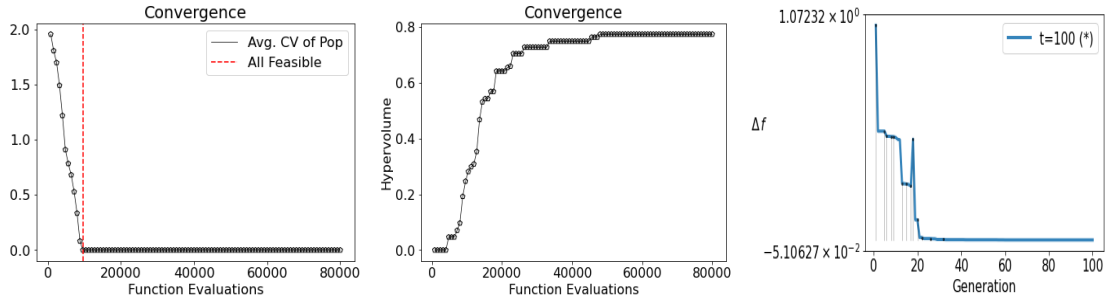
For CR, results with regards to convergence speed, best seen in subplot (b), are similar compared to the first search. Performance is best for $CR = 1$, and deteriorates when the value for CR decreases. When analysing the different planes displaying CR with respect to dot color, best seen in subplots (b) and (c), we see the best results in the plane $CR = 0.8$. We choose to focus on the plane $CR = 0.8$, as this value yields acceptable convergence speed in combination with best performance regarding hypervolume indicator.

Similar to the first grid search, F does not affect convergence speed of the algorithm. When analysing dot color in the plane $CR = 0.8$, best seen in subplots (b) and (c), we see equal performance for values between $F = 0.5$ and $F = 1.25$. Thus, the value of F should be set somewhere within this interval.

The last parameter to analyse is population size N, starting with dot size. Similarly to the first grid search, N does not affect convergence speed of the algorithm. If we analyse the plane $CR = 0.8$ with respect to dot color, best seen in subplots (b) and (c) we see that a decreasing value of N yields lower values for hypervolume indicator. If N is decreased to $N = 165 = 3D$, we see good performance in combination with $F = 0.5$ and $F = 1$. If N is decreased further, algorithm performance deteriorates. From the second gird search we can conclude that population size could be decreased to $3D$, in combination with $CR = 0.8$ and $F = 0.75$. However, convergence speed will suffer from a decreased population size.

C Solving problem instances with GDE3

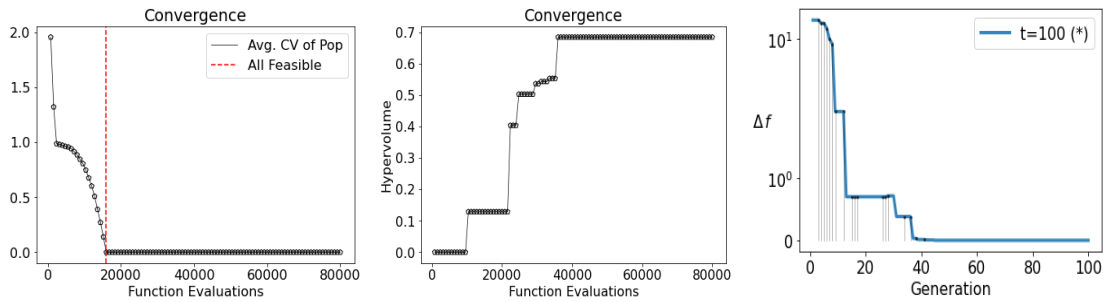
The plots below show the performance indicators resulting from running GDE3 for 100 generations with parameter settings $CR = 1$, $F = 1$ and $N = 800$ for the three different problem instances. When analysing the first problem instance, subplot (a) shows the convergence of population feasibility. After 9600 function evaluations, or 12 generations, all individuals within the population are feasible. Subplot (b) shows the convergence of the hypervolume indicator. After 47200 function evaluations, or 59 generations, the hypervolume indicator stopped converging and thus the algorithm reached an optimum. The running metric, depicted in subplot (c) shows that after approximately 30 generations, hypervolume indicator convergence is very limited. With computational time of the grid search in mind, it is decided to set the termination criterion to 50 generations. It is expected that GDE3 has converged to a near optimal solution before this point.



(a) Feasibility (b) Hypervolume indicator (c) Running metric

Figure 15: Performance measures 10 jobs, 100 generations, instance 1

Subplot (a) of the second problem instance shows the convergence of population feasibility. After 16000 function evaluations, or 20 generations, all individuals within the population are feasible. Subplot (b) shows the convergence of the hypervolume indicator. After 35200 function evaluations, or 44 generations, the hypervolume indicator stopped converging and thus the algorithm reached an optimum. The running metric, depicted in subplot (c) shows that after approximately 40 generations, hypervolume indicator convergence is very limited. With computational time of the grid search in mind, it is decided to set the termination criterion to 60 generations. It is expected that GDE3 has converged to a near optimal solution before this point.

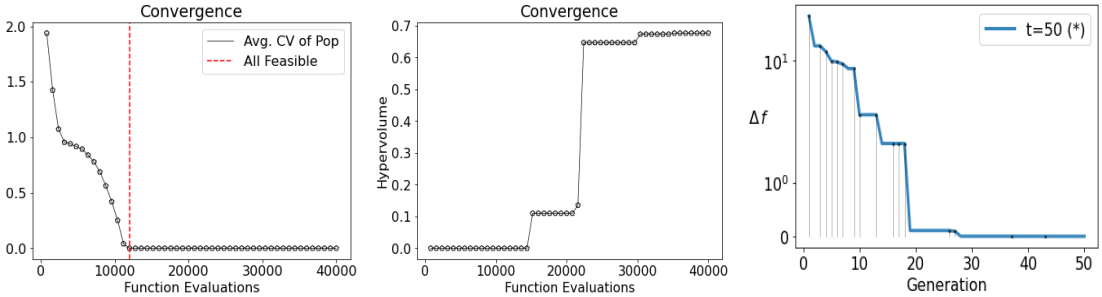


(a) Feasibility (b) Hypervolume indicator (c) Running metric

Figure 16: Performance measures 10 jobs, 100 generations, instance 2

Subplot (a) of the third problem instance shows the convergence of population feasibility. After 12000 function evaluations, or 15 generations, all individuals within the population are feasible. Subplot (b) shows the convergence of the hypervolume indicator. After 34400 func-

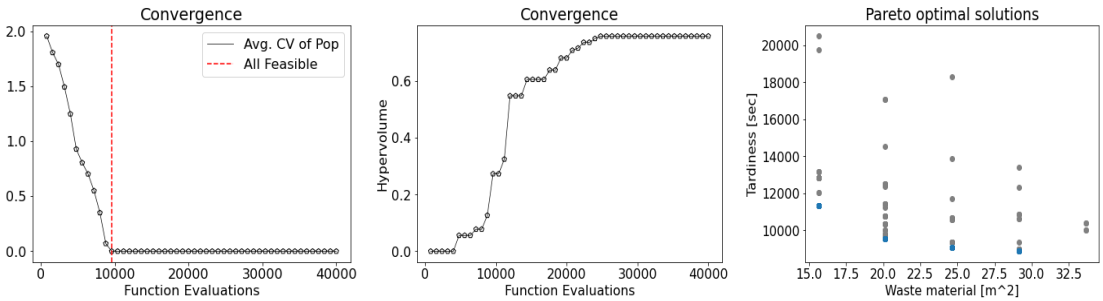
tion evaluations, or 43 generations, the hypervolume indicator stopped converging and thus the algorithm reached an optimum. The running metric, depicted in subplot (c) shows that after approximately 30 generations, hypervolume indicator convergence is very limited. With computational time of the grid search in mind, it is decided to set the termination criterion to 60 generations. It is expected that GDE3 has converged to a near optimal solution before this point.



(a) Feasibility (b) Hypervolume indicator (c) Running metric

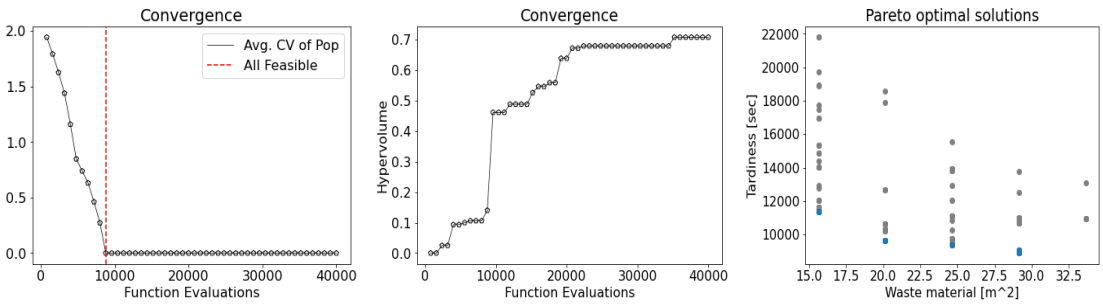
Figure 17: Performance measures 10 jobs, 100 generations, instance 3

For each of the problem instances, 5 runs are executed. Per run, a figure displaying the performance measures convergence speed with regards to population feasibility and hypervolume indicator is shown below. Next to the performance measures, the found Pareto set including feasible solutions found during the evolution of the algorithm are included as well.



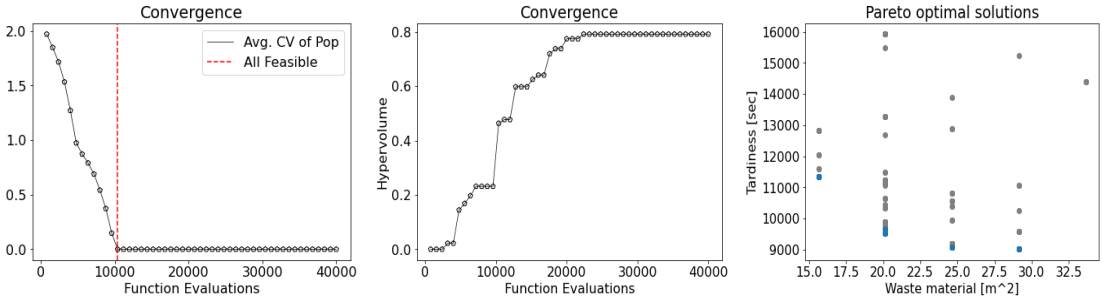
(a) Feasibility (b) Hypervolume indicator (c) Pareto set

Figure 18: Problem instance 1, run 1



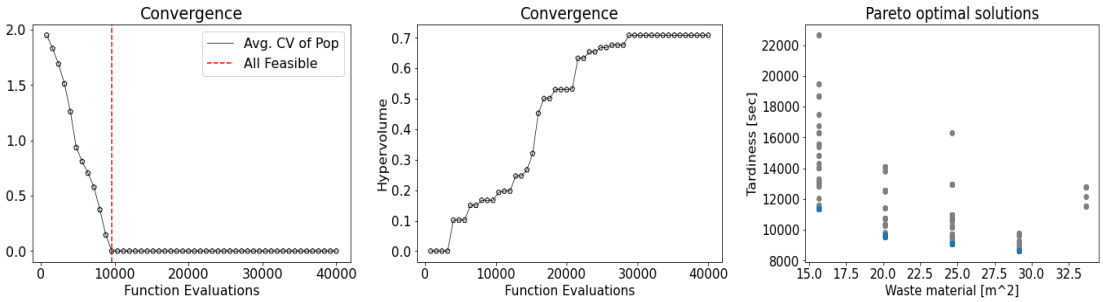
(a) Feasibility (b) Hypervolume indicator (c) Pareto set

Figure 19: Problem instance 1, run 2



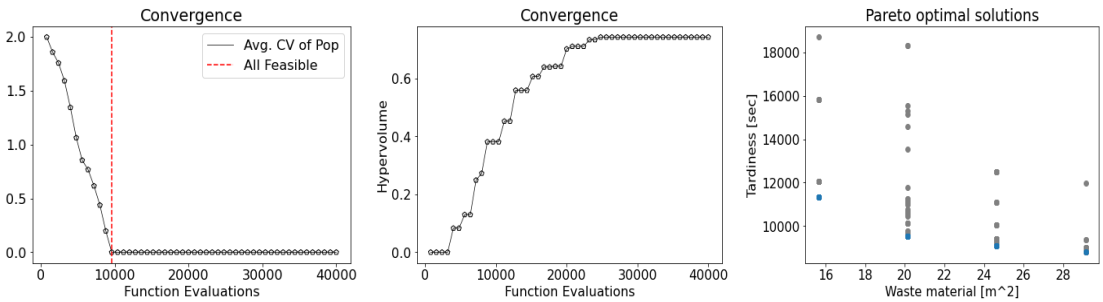
(a) Feasibility (b) Hypervolume indicator (c) Pareto set

Figure 20: Problem instance 1, run 3



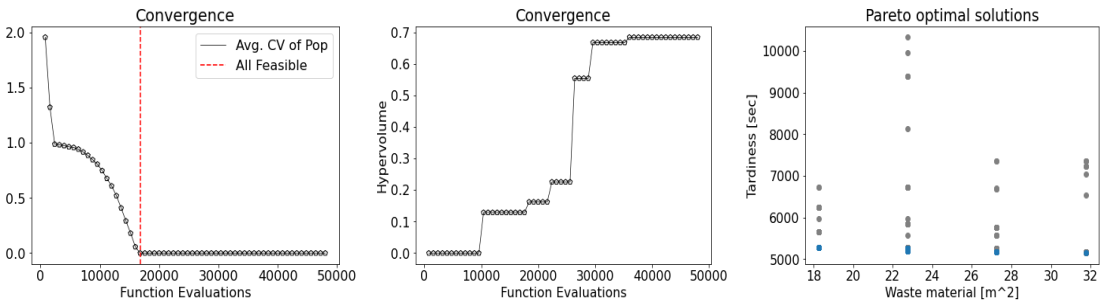
(a) Feasibility (b) Hypervolume indicator (c) Pareto set

Figure 21: Problem instance 1, run 4



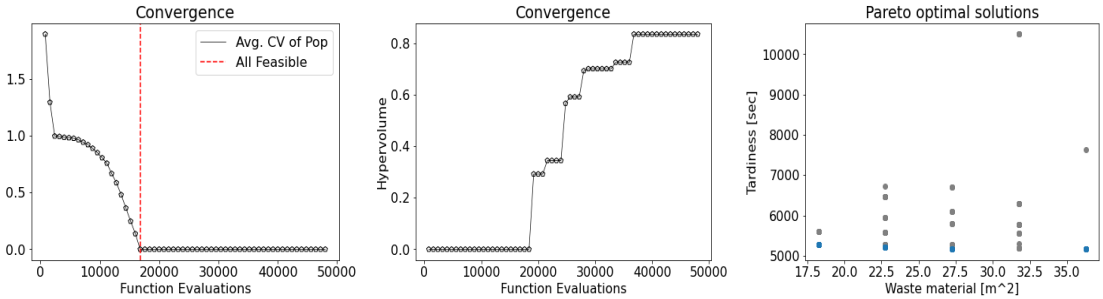
(a) Feasibility (b) Hypervolume indicator (c) Pareto set

Figure 22: Problem instance 1, run 5



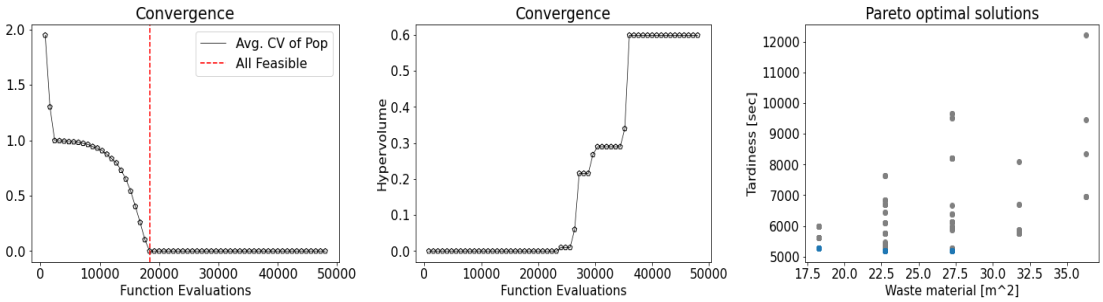
(a) Feasibility (b) Hypervolume indicator (c) Pareto set

Figure 23: Problem instance 2, run 1



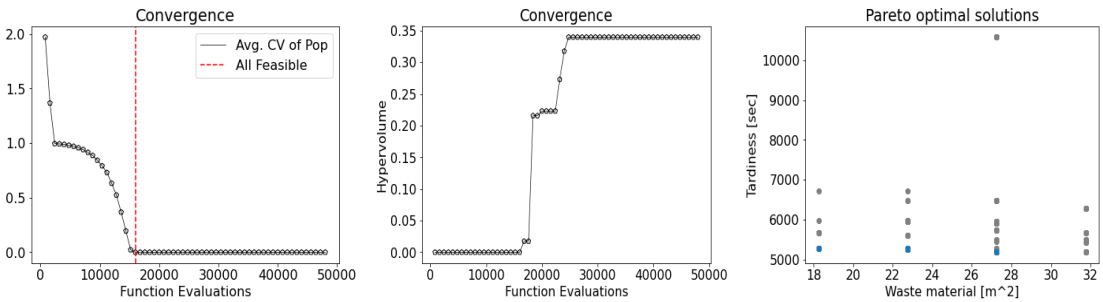
(a) Feasibility (b) Hypervolume indicator (c) Pareto set

Figure 24: Problem instance 2, run 2



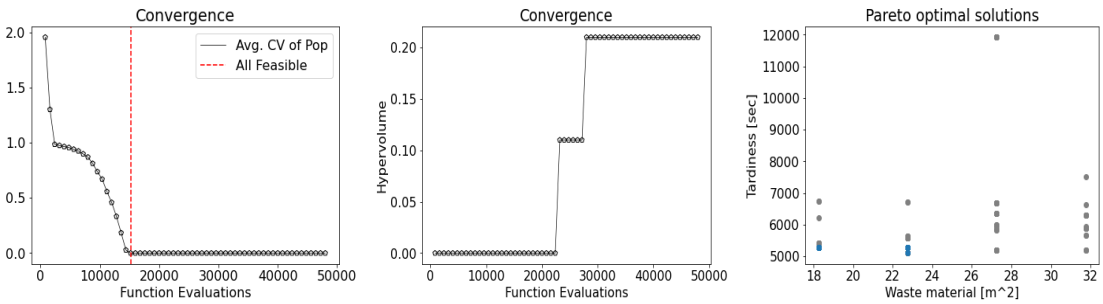
(a) Feasibility (b) Hypervolume indicator (c) Pareto set

Figure 25: Problem instance 2, run 3



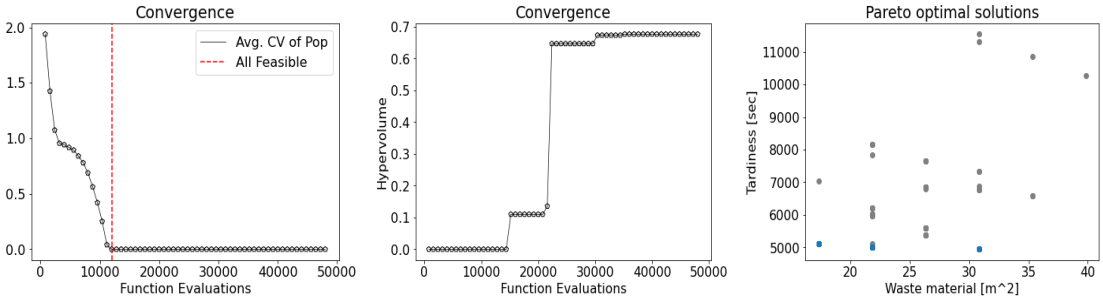
(a) Feasibility (b) Hypervolume indicator (c) Pareto set

Figure 26: Problem instance 2, run 4



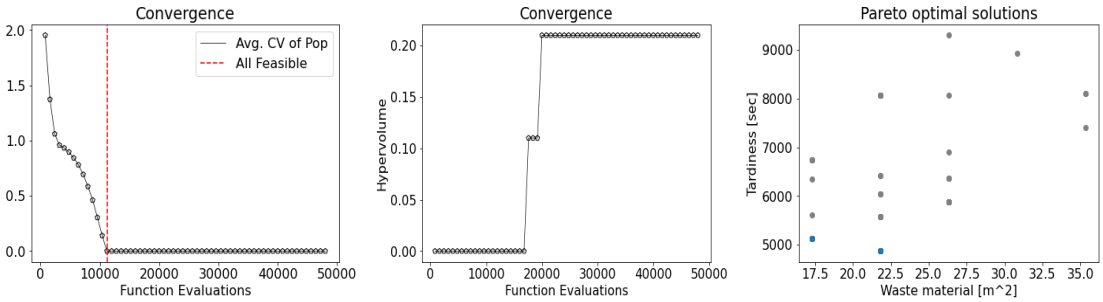
(a) Feasibility (b) Hypervolume indicator (c) Pareto set

Figure 27: Problem instance 2, run 5



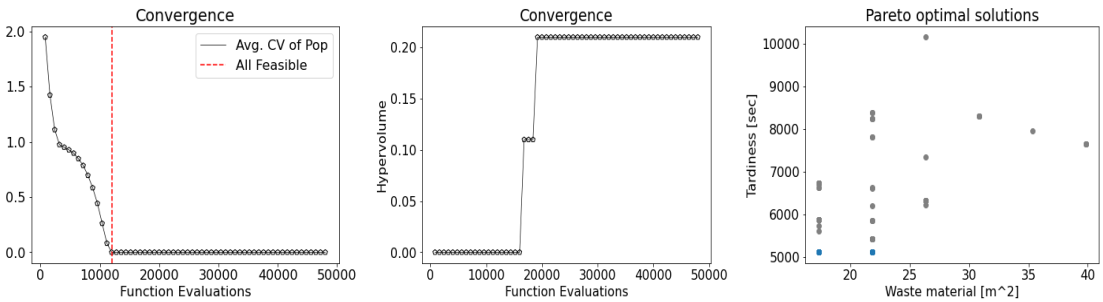
(a) Feasibility (b) Hypervolume indicator (c) Pareto set

Figure 28: Problem instance 3, run 1



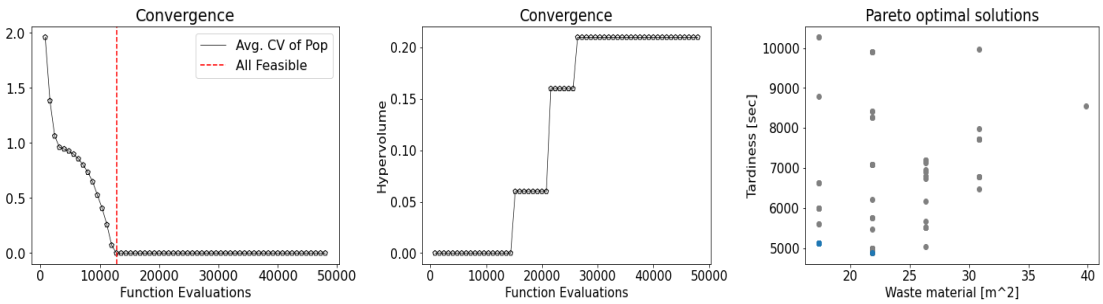
(a) Feasibility (b) Hypervolume indicator (c) Pareto set

Figure 29: Problem instance 3, run 2



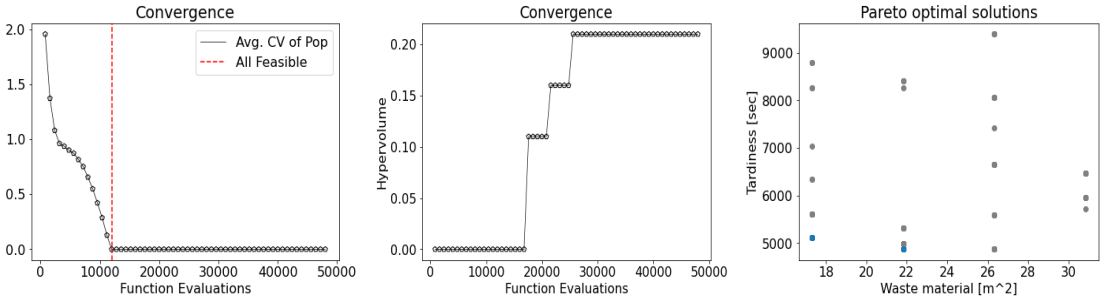
(a) Feasibility (b) Hypervolume indicator (c) Pareto set

Figure 30: Problem instance 3, run 3



(a) Feasibility (b) Hypervolume indicator (c) Pareto set

Figure 31: Problem instance 3, run 4



(a) Feasibility (b) Hypervolume indicator (c) Pareto set

Figure 32: Problem instance 3, run 5