Combining Heuristics and Machine Learning for Hybrid Flow Shop Scheduling Problems

Von der Fakultät für Ingenieurwissenschaften,

Abteilung Maschinenbau und Verfahrenstechnik der

Universität Duisburg-Essen

zur Erlangung des akademischen Grades

einer

Doktorin der Ingenieurwissenschaften

Dr.-Ing.

genehmigte Dissertation

von

M.Sc. Miriam Zacharias

aus

Duisburg

1. Gutachter:	Prof. Dr. rer. nat. Johannes Gottschling
2. Gutachter:	Prof. Dr.-Ing. Bernd Noche
Tag der Disputation:	11.08.2020

In Memoriam

Prof. Dr. Rainer Leisten

Abstract

This dissertation presents a new generic scheduling approach to makespan and flowtime minimization in hybrid flow shops with unrelated machines. In addition, possibilities of utilizing machine learning tools in job scheduling are explored. The proposed scheduling approach is twofold. First, a new heuristic based on the *divide et impera* strategy is introduced, that can be optionally enhanced by different local search improvement schemes. Moreover, several parameters are provided that enable a trade-off between solution quality and computation time. Then, a new hybrid variant of the heuristic is presented, that drastically reduces computation times through the application of machine learning techniques.

Initially, the formal problem definition is given in form of a mixed integer linear programming formulation, and, based on an extensive literature review, the research gap is outlined.

To evaluate the performance of the new approach, a testbed representing various production settings is introduced. Due to the NP-hardness of the problem, optimal solutions cannot be determined for all instances. Therefore, several lower bounds are proposed for makespan minimization. Additionally, the well-known heuristic of Nawaz et al. (1983) is considered as benchmark for makespan and flowtime, as well as solutions of the mathematical optimization model determined in a specified time limit.

A t-test for paired, dependent samples is conducted to statistically validate the results of the new approach. Furthermore, stability aspects of the results are evaluated and a study on computation times is included. Lastly, possible scope for future work is highlighted.

Kurzfassung

Die vorgelegte Dissertation stellt einen neuen generischen Planungsansatz zur Minimierung von Makespan und Flowtime in hybriden Flowshops mit nichtidentischen Maschinen vor. Darüber hinaus werden Möglichkeiten des Einsatzes von Methoden des maschinellen Lernens in der Maschinenbelegungsplanung untersucht. Der vorgeschlagene Planungsansatz ist zweigeteilt. Zunächst wird eine neue Heuristik auf Basis der *Divide-et-Impera*-Strategie eingeführt, die optional durch verschiedene lokale Suchheuristiken ergänzt werden kann. Zusätzlich verfügt die Heuristik über Parameter, die eine Steuerung des Kompromisses zwischen Lösungsqualität und Rechenzeit ermöglichen. Im zweiten Teil wird eine neue hybride Variante des heuristischen Lösungsansatzes vorgestellt, bei der durch den Einsatz von maschinellen Lerntechniken die Rechenzeiten drastisch reduziert werden kann.

Zunächst wird die formale Problembeschreibung in Form eines gemischt ganzzahligen linearen Optimierungsproblems dargestellt und, auf der Grundlage einer umfangreichen Literaturrecherche, die Forschungslücke skizziert.

Zur Bewertung des neuen Ansatzes wird ein repräsentatives Testbed eingeführt, welches verschiedene Produktionsumgebungen modelliert. Aufgrund der NP-Vollständigkeit des Problems können nicht für alle Instanzen optimale Lösungen gefunden werden. Daher werden verschiedene untere Schranken für die Makespan-Minimierung entwickelt. Darüber hinaus wird die etablierte Heuristik von Nawaz et. al. (1983) als Benchmark für Makespan und Flowtime verwendet. Zudem werden die Lösungen des mathematischen Optimierungsmodells mit vorgegebener Rechenzeit als Vergleichsgrößen herangezogen.

Zur statistischen Validierung der Ergebnisse des neuen Ansatzes wird ein t-Test für abhängige Stichproben durchgeführt. Weiterhin werden Stabilitätsaspekte der Ergebnisse evaluiert und eine Studie über die Rechenzeiten durchgeführt. Schließlich wird auf mögliche Forschungsfragen für zukünftige Arbeiten hingewiesen.

Contents

List of Tables

List of Figures

List of Abbreviations

ACO	ant colony optimization
BPN	feedforward network(s) with backpropagation learning
C4.5	decision tree generating algorithm
CPU	central processing unit
DR	dispatching rule(s)
DT	decision tree(s)
eSRPT	extended shortest remaining processing time
FMS	flexible manufacturing system(s)
FS	flow shop(s)
GA	genetic algorithm(s)
gSPT	generalized shortest processing time
GUI	graphical user interface
HFS	hybrid flow shop(s)
HNN	Hopfield neural network(s)
ID3	decision tree generating algorithm
LB	lower bound(s)
LPT	longest processing time
LS	local search
MAJ	modular assignment of jobs
MILP	mixed integer linear programming
ML	machine learning
NEH	heuristic of Nawaz, Enscore and Ham
NN	neural network(s)
NP	nondeterministic polynomial time
P	polynomial time
RRMSE	relative root mean square error
SA	simulated annealing
SMAPE	symmetric bean absolute percentage error
SOM	self-organizing map(s)

SPT	shortest processing time
SVM	support vector machine(s)
SVR	support vector regression
TSP	travelling salesman problem(s)

List of Symbols

Chapter 2:

α	machine environment
α_1	general configuration as HFS
α_2	number of stages of HFS
α_3	HFS machine property: type
α_4	HFS machine property: no. of machines
β	scheduling constraints
γ	objective function
batch	batch or grouping constraints
block	blocking constraint
$\overline{C}$	total/average completion time
C_j	completion time of job j
$C_{\max}$	makespan
$\overline{C}^w$	total/average weighted completion time
$\overline{E}$	total/average earliness
E_j	earliness of job j
$E_{\max}$	maximum earliness
$\overline{E}^w$	total/average weighted earliness
$\overline{F}$	total/average flowtime
F_j	flowtime of job j
F_m	flow shop with m machines
$F_{\max}$	maximum flowtime
$\overline{F}^w$	total/average weighted flowtime
FH_m	hybrid flow shop with m stages
FJ_c	flexible job shop with c work centres
J	set of jobs
J_m	job shop with m machines
L_j	lateness of job j
$L_{\max}$	maximum lateness

lag	overlap constraint
lim-wait	limited waiting time constraint
M_j	machine eligibility constraints
nJ	total number of jobs
no-wait	no waiting time constraint
O_m	open shop with m machines
P	identical machines
P_m	parallel machines Problem with m identical machines
prec	precedence constraint
prmp	pre-emption allowance
prmu	permutation constraint
R	unrelated machines
R_m	parallel machines Problem with m unrelated machines
rc	resource constraint
rcrc	recirculation constraint
r_j	release date of job or machine j
size_{jk}	simultaneous processing option
skip	skipping of stages option
S_{sd}	sequence dependent setup times
S_{si}	sequence independent setup times
stoch	stochastic parameters
$\overline{T}$	total/average tardiness
T_j	tardiness of job j
$T_{\max}$	maximum tardiness
$\overline{T}^w$	total/average weighted tardiness
unavail	machine availability constraint
U	uniform machines
U_m	parallel machines Problem with m uniform machines
$\overline{U}$	number of late jobs
U_j	binary variable to indicate if job j is late
$\overline{U}^w$	total weighted number of late jobs

Chapter 3:

l_i	number of machines at stage i
M^i	set of machines at stage i

Chapter 4:

x	feature vector
x_n	input data instance
y_n	output data instance

Chapter 5:

$\mathsf{big}M$	big number
$C_{j,i}$	completion time of job j on stage i
C_{max}	makespan
$CS_{k,i}$	completion time of job in position k on stage i
$\overline{F}$	flowtime
I	set of stages
i	stage index, $i = 1, ..., n_I$
J	set of jobs
j	job index, $j = 1, ..., n_J$
k	position index, $k = 1, ..., n_J$
L	set of machines
l	machine index, $k = 1, ..., n_L$
M^i	set of machines at stage i
n_I	number of stages in I
n_J	number of jobs in J
n_L	total number of machines in L
P_{jl}	processing times of job j on machine l
$S_{k,i}$	start time of job in position k on stage i
$T_{j,i}$	start time of job j on stage i
$X_{j,k}$	binary decision variable for job positions
$Y_{i,k,l}$	binary decision variable for machine assignments

Chapter 6:

α_i	minimal front waiting time of machine i
β_i	minimal back idle time of machine i
γ_i	total workload of machine i
a_i	availability time of machine i
A_k	lower bound on total front waiting time of stage k
A_{hk}	lower bound on total front waiting time between stages h and k
B_k	lower bound on total back idle time of stage k
B_{kl}	lower bound on total back idle time between stages k and l
$C^*_{\max}$	optimal makespan
eSPT_k	vector of jobs' completion times of the eSPT-rule solution for the parallel machine problem at stage k
$\text{eSPT}_k(m_k)$	sum of completion times of the first m_k jobs with minimal completion times of eSPT-rule solution
gSPT_k	vector of jobs' completion times of the gSPT-rule solution for the parallel machine problem at stage k
$\text{gSPT}_k(m_k)$	sum of completion times of the first m_k jobs with minimal completion times of gSPT-rule solution
h, k, l	stage indices
i	machine index
j	job index
LB	lower bound
LS-values	left-side total processing times of all jobs before stage k
LSA_i	i-th smallest value of LSA-list
LSA-list	list of sorted LS-values
m_k	no. of identical machines at stage k
M^k	set of identical machines at stage k
NC_{inc}	number of machines non-decreasing with time
NEH^{HFS}	NEH-method modified for Hybrid Flow Shop problems
p_{ji}	processing time of job j on machine i
$p_{[j]i}$	j-th smallest processing time on machine i

$q_{k[i]}$	i-th smallest value of all q_{kj}-values
q_{kj}	right-side total processing times of job j after stage k
R^2	coefficient of determination
r_j	release date of job j
$r_{k[i]}$	i-th smallest value of all r_{kj}-values
r_{kj}	left-side total processing times of job j before stage k
RS-values	right-side total processing times of all jobs after stage k
RSA_i	i-th smallest value of RSA-list
RSA-list	list of sorted RS-values
SPT_k	vector of jobs' completion times of the SPT-rule solution for the parallel machine problem at stage k
$\text{SPT}_k(m_k)$	sum of completion times of the first m_k jobs with minimal completion times of SPT-rule solution

1 Introduction

Sequencing and scheduling is crucial in manufacturing and service industries as a regularly occurring form of decision-making. Scheduling plays an important role in most manufacturing and production systems as well as in most information processing environments, in transportation and distribution settings, and in other types of service industries. Effective operations sequencing has become a necessity for companies in the current competitive environment, which e.g. have to meet customer delivery dates that have been committed to, as otherwise a significant loss of reputation and monetary penalties might occur. Also, efficient resource utilization through scheduling helps to optimize costs. (Pinedo (2016))

Scheduling deals with the allocation of resources of many different forms to various sets and types of tasks, being parts of some processes, over given time periods and its goal is to optimize one or more objectives. Tasks individually compete for resources which can be of a very different nature, e.g. money, processors/machines, energy, tools, transport units or routes (like runways at an airport), manpower (like crews at a construction site), or processing units in a computing environment, and the tasks may e.g. be operations in a production process, logistic operations (like take-offs and landings at an airport), stages in a construction project, or executions of computer programs. (Pinedo (2016), Blazewicz et al. (2007))

The same holds for task characteristics and constraints that might have to be taken into consideration, e.g. ready times, due dates, relative urgency weights, precedence relations, or functions describing task processing in relation to assigned resources. In addition, different criteria to evaluate the quality of the performance of a sequenced set of tasks can be taken into account. Those may vary largely depending on the scheduling environment. Objectives can be completion time or due date related but may also be aiming at e.g. a smooth process flow or utilization optimization. (Blazewicz et al. (2007))

Based on this general definition, scheduling problems occur almost everywhere

in real-world situations. Many aspects concerning approaches for modelling and solving these problems have been presented during the last decades that are of general methodological importance. On the other hand, some classes of scheduling problems have their own specificity which should be taken into account while developing suitable solution approaches. Scheduling system design and development is predominantly performed by computer scientists, operations researchers, and industrial engineers. (Blazewicz et al. (2007))

The interdisciplinary nature of the scheduling research community led to many metaheuristic adaptations and hybrid approaches with different streams of focus areas. Consequently, machine learning became part of the research interest and new scheduling methods inspired by artificial intelligence approaches were sought.

Machine learning algorithms, as a subset of artificial intelligence in the field of computer science, are aiming at performing tasks by generalizing from examples without being manually programmed. Progressive digitization and the availability of more and more data allows more complex problems to be addressed. (Domingos (2012))

Another aspect of machine learning focusses on how to optimize a performance criterion based on example data or past experience. A model is defined up to some parameters and a computer program is executed to learn optimal parameters of the model using training data or past experience. (Alpaydin (2009))

According to Witten et al. (2016), this learning process is defined as the search through an enormous but finite search space of possible rules to predict output based on data. This means that based on given data, an algorithm for prediction of the desired output is learned. Generalization being the goal has the consequence that, unlike in most other optimization problems, the objective function is not necessarily analytic and training error has to be used as a surrogate for test error. (Domingos (2012))

With regards to scheduling problems, the discrete nature and the complexity of such optimization problems proved to be challenging for pure machine learning approaches. Nevertheless, integrating machine learning techniques

into scheduling algorithms yielded promising results, and exploring options to utilize the advantages of machine learning tools in scheduling decisions became an important part of the current research.

This study focusses on the hybrid flow shop scheduling problem with non-identical machines, a setting that often occurs in real-life production environments, but that is known to be difficult to solve. (Ruiz and Vázquez-Rodríguez (2010)) We present a new generic approach that is able to deal with different variants of hybrid flow shops with regards to stage configurations and objectives. In a second step, the utilization of machine learning techniques is explored.

The structure of this thesis is as follows: In section 3, a broad introduction to basic scheduling concepts is given, starting with an overview of classes of scheduling problems in general. Then, the notation used throughout this work, which was developed by Graham et al. (1979) and extended by Vignier et al. (1999), is introduced. This notation clearly identifies the shop configuration, relevant constraints, and the objective function for a given scheduling problem. Typical examples for each feature of such a scheduling problem are presented. Subsequent to the introduction to scheduling classes, an overview of solution methods is provided, categorised as exact algorithms and approximate algorithms, focussing on heuristics and metaheuristics.

Section 4 specifies the hybrid flow shop scheduling problem in detail, starting with the standard case and highlighting complexity considerations. The section is concluded by a comprehensive overview on the state of the art in research, focussing on the hybrid flow shop with unrelated machines.

As the hybrid approach proposed later on includes a machine learning model, an introduction to machine learning in general is presented in section 5, starting with a description of the basic concepts and followed by a summary of selected machine learning algorithms. The section concludes with an exhaustive presentation of the work to date by researchers applying machine learning techniques to scheduling problems.

Based on the discussions in sections 4 and 5, a research gap becomes apparent

with regards to flexible approaches for hybrid flow shops with unrelated machines as well as regarding the utilization of machine learning tools in hybrid flow shop scheduling. The consequential formal problem formulation is then presented in the form of a mathematical model in the first part of section 6. After the scope of the research is clearly defined, a new testbed is introduced, representing various production processes following the approach of Watson et al. (2002).

As no benchmark solutions can be taken from the literature, upper and lower bounds are defined in a next step. For makespan minimization, existing lower bounds for the hybrid flow shop with identical machines can be adapted to deal with the case of unrelated machines. For both objectives considered, i.e. makespan and flowtime, upper bounds are generated by a modified version of the well-known heuristic proposed by Nawaz et al. (1983). In addition, a mixed integer linear programming model is used to generate solutions within a given computation time for comparison purpose. Finally, the proposed twofold scheduling approach is presented in the sections 6.2 and 6.3.

First, the new heuristic based on the *divide et impera* strategy is introduced and the optional different local search improvement schemes are presented in detail. Moreover, several parameters are provided that enable a trade-off between solution quality and computation time.

Then, the new hybrid variant of the heuristic is described, which includes machine learning techniques to reduce computation times. The machine learning model is explained in detail and for both heuristics, all major algorithm steps are presented as pseudocode.

The results of the numerical experiments are systematically reviewed in section 7. At first, the proposed heuristic without machine learning is evaluated with regards to the influence of parameter settings and the success of local search improvement schemes. The performance, compared to that of the heuristic by Nawaz et al. (1983), is statistically validated by a t-test for dependent samples. Afterwards, the usability of the machine learning model is tested and the results are evaluated focussing on the savings in computation

time. The section is concluded by an extensive study on the stability of the new approach, meaning that the effect of changes in the input data on the objective function values determined by the scheduling heuristic is investigated. This dissertation is completed by the conclusion given in section 8 and highlights the scope for future work in the outlook summarized in section 9.

2 Classification of Scheduling Problems

The allocation of limited resources to tasks over time is the quintessence of scheduling theory. It is a decision-making process that aims at the optimization of one or more objectives. (Pinedo (2016))
As market demands for product quality, flexibility and order flowtimes are increasing continuously, the need for efficient scheduling has simultaneously increased in the last decades. A plethora of parameters is to be taken into account solving scheduling problems. The nature of resources can vary (e.g. manpower, money, machines, and tools) as well as the task characteristics (e.g. ready times, due dates, relative urgency weights, and functions of task processing). Furthermore, different ways to model structures of a set of tasks occur, which reflect the dependencies among them. Moreover, the performance measurement criteria can differ depending on the nature of the problem. (Blazewicz et al. (2007))
Scheduling problems are combinatorial optimization problems and most decision variables are discrete in nature. Traditionally there are three kinds of regular objective functions considered in job scheduling, namely utilization-orientated (such as makespan), material flow-orientated (such as flowtime) and due date-orientated functions (such as earliness, lateness and tardiness). In addition, since the 1970s homogeneity-orientated and irregular functions arose (e.g. completion time variance). (Leisten and Rajendran (2015))

2.1 Problem Classes

Scheduling problems can be classified as static versus dynamic and deterministic versus stochastic problems. If all information about the planning horizon is complete at the time of planning, that means all characteristics of a set of tasks and a set of resources are known, the problem is classified as static. In contrast, in dynamic problems some parameters are unknown in advance. In deterministic problems, no variable with non-deterministic (e.g. probabilistic) description appears. (Blazewicz et al. (2007))

If such variables occur, the problem is a stochastic scheduling problem where decisions without full information about their outcomes are required. (Blazewicz et al. (2007))
Most scheduling decisions are based on static-deterministic approaches while state of the art in the related field of queuing theory is a dynamic and stochastic perspective. (Leisten and Rajendran (2015))
In this work, we consider a static deterministic scheduling problem.

2.2 Shop Configuration

Graham and Vignier Notation

Graham et al. (1979) introduce a notation scheme for standard theoretic scheduling problems based on a triplet $\alpha|\beta|\gamma$ to characterize the basic features. In this context α defines the machine environment (or shop configuration), whereas β includes information about job characteristics and constraints and γ denotes the objective function.
Since its introduction, the Graham notation has been constantly extended to describe some varieties of standard problems in more detail. An example for a widely used extension is the notation of Vignier et al. (1999) for the special case of hybrid flow shop scheduling problems. Parameter α is split into a quadruple to be able to adequately describe the multi-stage setting of this shop configuration.
This thesis focusses on a common manufacturing environment, the hybrid flow shop, which will be characterized after a short overview of standard scheduling problems it is based on. The introduction to basic concepts of scheduling in the following sections is based on Pinedo (2016) if not stated otherwise.

Single Machine

The most simple of all machine environments is the single machine ($\alpha = 1$) case. It is a special case of all other more complicated environments and for some objective functions (like makespan) it is trivial to find an optimal solu-

tion. Nevertheless, there are many objectives (e.g. minimizing the maximum lateness with given release and due dates) for that even the single machine case is proven to be NP-complete[1]

Parallel Machines

Parallel Machine cases can differ with respect to the processing times of jobs. In case $\alpha = P_m$ there are m identical machines in parallel. Without further specification, a job requires a single operation and may be processed on any of the m machines.

Machines in parallel with different speeds are denoted by $\alpha = U_m$. The processing time of a job on one of those machines is equal to the standard processing time of the job divided by the machine's speed (assuming the job receives all its processing from one machine). This environment is referred to as uniform machines. A further generalization of the previous environment is the case of unrelated parallel machines ($\alpha = R_m$), where the speed of a machine additionally depends on the job being processed.

The simple case of makespan minimization on two identical machines is already NP-complete. (Lenstra et al. (1977))

Flow Shop

If machines are aligned in series, the environment is classified as a flow shop ($\alpha = F_m$). There are m machines (also called stages) in series and each job has to be processed on each one of the m machines. All jobs follow the same route and have to be processed first on machine 1, then on machine 2, and so on until they are completed on machine m. A common assumption is that the sequence of jobs is the same on all machines and the β-field contains the entry prmu for this case, representing a permutation flow shop. Keeping

[1] We denote by P the class of problems for which a polynomial-bounded (good or efficient) algorithm exists, and by NP the class of problems which can be solved by polynomial-depth backtrack search (see Lenstra et al. (1977)). Following a definition by Lawler et al. (1993), we call a problem NP-complete if its solvability in polynomial time would imply that every problem in NP is solvable in polynomial time as well, i.e. that P = NP. (Lenstra et al. (1977))

the sequence of jobs fixed equals a first in, first out rule for queues between machines. Minimizing the total completion time of jobs in a two-stage flow shop is already NP-complete. (Lenstra et al. (1977))

Job Shop

Job Shops ($\alpha = \mathrm{J}_m$) are machine environments where each job has its own predetermined route to follow. A distinction is made between Job Shops in which each job visits each of the m machines at most once and Job Shops with recirculation (β-field = rcrc), in which a job may visit each machine more than once. An extension of the classical job shop is the flexible job shop, denoted by $\alpha = \mathrm{FJ}_c$, which is a combination of job shop and parallel machine environments. Instead of m machines there are c work centres with identical machines in parallel and each job requires processing on one machine at each work centre following its respective route. Open Shops are the most flexible Job Shop setting ($\alpha = \mathrm{O}_m$). Each job has to be processed on one of the m machines but some of these processing times may be zero and there are no restrictions with regard to the routing of each job through the machine environment. Job shop problems with only two machines and makespan objective are proven to be NP-complete. (Garey et al. (1976))

Hybrid Flow Shop

Hybrid flow shops ($\alpha_1 = \mathrm{FH}_m$) (also referred to as flexible flow shops, multi-processor flow shops or flexible flow lines) are a generalization of the flow shop and the parallel machine environments. Instead of m machines there are i stages in series, each containing a subset of M^i parallel machines. Each job has to be processed first at stage 1, then at stage 2, and so on and at each stage jobs require processing on only one machine. The sequence of jobs may or may not be the same on all stages.

Following Vignier et al. (1999), parameter α is split into $\alpha_1, ..., \alpha_4$ and defines the structure of the shop. α_1 indicates the general configuration of the shop as HFS, α_2 is the number of stages, α_3 and α_4, in combination, describe the

properties of machines per stage. The notation $(\alpha_3 * \alpha_4)^i$ states that there are α_4 parallel machines of the type α_3 in stage i. $\alpha_3 = P$ indicates identical parallel machines, $\alpha_3 = U$ uniform parallel machines and $\alpha_3 = R$ unrelated parallel machines. In the case of a single machine at a stage, α_3 is empty. In total, α_3 and α_4 have to be repeated for every stage specified in α_2.
Most HFS problem variants are proven to be NP-complete. For example, Gupta (1988) show that makespan minimization in a two-stage HFS, even in the case when one stage contains two identical machines and the other one a single machine, is NP-complete.

Constraints

A plethora of constraints and specifications have to be considered in practical scheduling scenarios. For theoretical models, a selection of constraints and assumptions including their respective β-field entries are summarized in table 2.2.
Usually, only a small number of additional constraints is considered when dealing with (F_m) and (FH_m) problems in theory. As mentioned before, even problems of most simple shop configurations are hard to solve so that only very small instances can be considered seeking an optimal solution. Additional constraints usually make the underlying model more complex and obtaining good solutions becomes more difficult to achieve.

Table 2.1: Constraints and assumptions in scheduling.

β-field entry	Name	Description
batch	batch or grouping constraints	Different jobs may be processed in parallel (batch) on some machines or grouping requirements of certain jobs might occur (e.g. in assembly systems).
block	blocking	In case of limited buffers between consecutive stages, upstream machines are not allowed to release a completed job if the buffer space in front of the next machine is fully occupied. The completed job has to remain on the upstream machine blocking that machine from processing the next job.
lag	overlap	Jobs are allowed to start being processed on the succeeding machine earlier than their completion time on the current machine (overlap of processing).
lim-wait	limited waiting time requirement	Jobs are not allowed to wait more than a given amount of time between two consecutive machines (implies delay of starting times of jobs at the first machine to ensure continuous processing throughout the system.
M_j	machine eligibility restrictions	Not all machines are capable of processing all jobs. Set M_j denotes the set of machines that can process job j.
no-wait	no waiting time requirement	Jobs are not allowed to wait between two consecutive machines (implies delay of starting times of jobs at the first machine to ensure continuous processing throughout the system.
prec	precedence constraints	One or more jobs may have to be completed before another job is allowed to start its processing. Special forms: Each job has at most one predecessor and at most one successor (chains), each job has at most one successor (intree), each job has at most one predecessor (outtree).
prmp	pre-emptions	Interruption of processing of a job is possible at any point in time. The amount of processing a pre-empted job already has received is not lost.
prmu	permutation	Order (or permutation) in which the jobs go through the first machine is maintained throughout a flow shop system.

β-field entry	Name	Description
rc	resource constraints	Machines might require renewable or consumable additional resources to process jobs. Such resources might include workers, transport systems or additional tools.
rcrc	recirculation	Jobs may visit a machine or work centre more than once (including re-work due to quality control).
r_j	release dates	Job [or machine] j cannot start its processing before its release date r_j. If r_j is not stated in β-field, all jobs might start at $r_j = 0$.
size$_{jk}$	simultaneous processing	Job j can or must be processed on size$_{jk}$ machines simultaneously at stage k (also called lot-splitting).
skip	skipping of stages	One or more jobs may not need to be processed on all stages.
S_{sd}/S_{si}	sequence dependent/independent setup times	Setup time that is incurred between the processing of consecutive jobs. Setup times can also be machine dependent or defined as clean-up times occurring after jobs have been processed.
stoch	stochastic parameters	Stochastic events might occur (e.g. stochastic job processing times or dynamic job arrivals).
unavail	machine breakdowns	Machines may not be continuously available (e.g. , due to shifts or scheduled maintenance).

Objective Functions

The γ-field denotes one or more objective functions to be considered. These can be completion time-oriented (e.g. makespan or flowtime), due date-oriented (e.g. lateness or earliness) as well as variability-oriented measures (e.g. completion time variance), but the objective to be minimized is always a function of completion times of jobs, which depend on the schedule.[2]

Regular objectives will improve if jobs can be completed earlier, that means a regular objective is a function that is non-decreasing in jobs' completion times. This relation does not hold for non-regular objectives like variability-related

[2] A distinction made between a sequence and a schedule is that a sequence usually corresponds to a permutation of jobs whereas a schedule is a timetable that includes all information about job's starting and completion times.

measures or objectives including earliness.[3]

Furthermore, it can be distinguished between Min-Max- and Min-Sum-objectives. Min-Max-objectives are aiming at minimizing a maximum value (e.g. the maximum lateness) while Min-Sum-objectives are used if the quality of a schedule depends on the completion time of all jobs.

The most commonly used objective in theory is minimization of makespan ($C_{\max}$), defined as the completion time of the last job to leave the system, which equals the maximum of the job's completion times C_j of all jobs $j \in J$, where J represents the set of jobs whereby each job is denoted by a natural number between 1 and $nJ := |J|$. A minimum makespan is assumed to achieve a high machine utilization. Although widely studied in research, makespan minimization is less important in real-world systems, where inventory cost or meeting customer demands might be more important, especially considering the fact that real manufacturing systems have to consider dynamic approaches. (Ruiz and Vázquez-Rodríguez (2010))

Other frequently-used objectives are variants of flowtime-measures, where a job's flowtime is defined as the time it spends in the system, i.e. the difference between it's completion time and release date. Jobs' flowtimes are summed up and might be multiplied with a weight factor to give a higher priority to certain jobs. An overview of common objectives in theory is given in table 2.2. Framinan et al. (2014) give a more detailed overview on scheduling problems from a theoretical and practical perspective. Figure 2.1 shows a selection of objectives by categories.

In this thesis we consider makespan and total flowtime as objective functions. As we do not specify release dates, the total flowtime equals the sum of completion times of all jobs.

[3]This is the reason why variability-related objectives are rarely considered as single objectives and rather included in multi-objective approaches. (See Leisten and Rajendran (2015) for some interesting findings on variability in scheduling).

Table 2.2: Common objective functions of scheduling problems. (Ruiz and Vázquez-Rodríguez (2010))

Notation	Description	Meaning
$C_{\max}$	$\max_{j \in J} C_{\mathrm{j}}$	Maximum completion time
$F_{\max}$	$\max_{j \in J}(C_j - r_j)$	Maximum flowtime
$L_{\max}$	$\max_{j \in J} L_j$	Maximum lateness
$T_{\max}$	$\max_{j \in J} T_j$	Maximum tardiness
$E_{\max}$	$\max_{j \in J} E_j$	Maximum earliness
$\overline{C}$	$\sum_{j=1}^{nJ} C_j$	Total* completion time
$\overline{C}^w$	$\sum_{j=1}^{nJ} w_j C_j$	Total* weighted completion time
$\overline{F}$	$\sum_{j \in J} F_j$	Total* flowtime
$\overline{F}^w$	$\sum_{j=1}^{nJ} w_j F_j$	Total* weighted flowtime
$\overline{T}$	$\sum_{j=1}^{nJ} T_j$	Total* tardiness
$\overline{T}^w$	$\sum_{j=1}^{nJ} w_j T_j$	Total* weighted tardiness
$\overline{U}$	$\sum_{j=1}^{nJ} U_j$	Number of late jobs
$\overline{U}^w$	$\sum_{j=1}^{nJ} w_j U_j$	Total* weighted number of late jobs
$\overline{E}$	$\sum_{j=1}^{nJ} E_j$	Total* earliness
$\overline{E}^w$	$\sum_{j=1}^{nJ} w_j E_j$	Total* weighted earliness

*Same symbols are also commonly used for the average, where the formulas in the description have to be multiplied by $1/nJ$.

2.3 Solution Methods

Scheduling problems are mainly NP-complete so that it is usually impossible to find optimal solutions, at least considering reasonable computation time. For some basic problems, exact algorithms exist to determine an optimal solution in polynomial time. For the majority of cases though, heuristic methods are applied to derive feasible, adequate or good quality solutions in reasonable time. The quality of such heuristic solutions can be evaluated by comparison with other benchmark heuristics or by estimating lower and upper bounds. Such bounds are based on the underlying scheduling problem, where defining a good bound itself is a complicated task.

If not stated otherwise, explanations of solution methods in the following sections are based on Framinan et al. (2014).

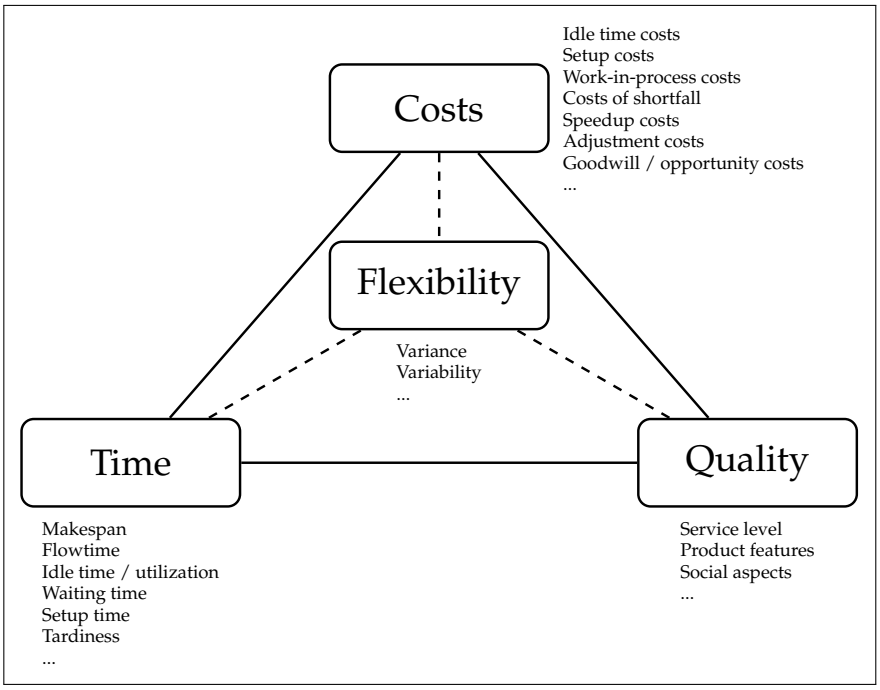


Figure 2.1: Common objective functions in theory and praxis. (Framinan et al. (2014)

An overview on classes of solution approaches for scheduling problems is given in figure 2.2.

Exact Algorithms

Exact algorithms, in contrast to approximate algorithms, guarantee that no other schedule leads to a better objective function value than the one obtained.

A distinction can be made between constructive and enumerate algorithms. Constructive methods exploit a specific model's properties to construct a guaranteed optimal solution. Although constructive procedures cover only very simple scheduling problems, they can be applied in approximate procedures based on decomposition techniques to solve sub-problems of more complex problems. For example, in praxis it could be sufficient to focus on an optimal solution for a specific machine in a production system if this machine is identified as the bottleneck[4] of the system.

[4]The bottleneck stage is defined as the stage with the least maximal possible throughput per time unit.

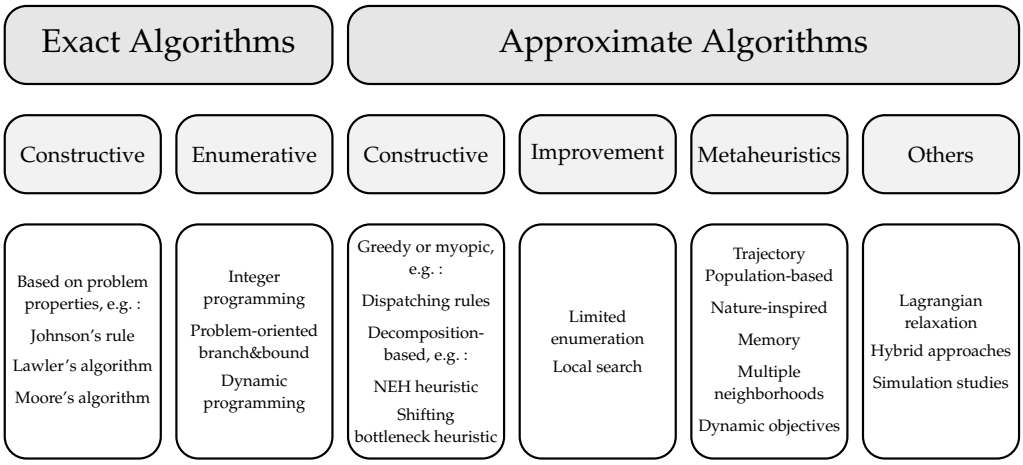


Figure 2.2: Solution approaches for scheduling problems. (Framinan et al. (2014)

Enumerative algorithms guarantee optimality by implicitly or explicitly evaluating all possible solutions of a model. Examples are complete enumeration, branch and bound, cutting plane or branch and cut approaches. Scheduling problems are often formulated as (Mixed) Integer Linear Programming Problems (MILP) so that the whole variety of branch and bound-methods can be utilized by applying Integer Programming Software to the generated model. Dynamic Programming algorithms decompose a problem into a series of overlapping sub-problems and recursively build up solutions to larger and larger sub-problems until the overall optimal solution is found. All enumerative procedures are extremely time-consuming and therefore large problems are usually intractable.

Heuristics

Heuristics are specifically tailored for a particular decision problem. Heuristics can be developed based on Relaxation, Decomposition, Adaptation or Reversibility. Relaxation means that the model under consideration is simplified until the remaining formal problem becomes tractable. This might lead to infeasible solutions so that an adjustment might be required in a second step. Decomposition procedures divide the problem into tractable sub-problems and coordinate the partial solutions to receive an integrated solution. Another idea is to adapt a well-known approach to a more complex or general problem.

Furthermore, traditionally scheduling procedures are forward-oriented but, in some context, it might be useful to start with the final assignment (last job on last machine) instead, following a backward-oriented perspective.
Constructive heuristics provide a first sequence or schedule exclusively based on the input data of the problem and construction rules which are supposed to return good or acceptable schedules (e.g. classical dispatching rules like shortest processing time rule). Many constructive procedures start with a sorted list of jobs and iteratively choose jobs from that list and add them to a so far generated partial schedule in a greedy or myopic way. A well-known and widely used constructive heuristic for the $Fm|prmu|C_{\max}$ problem is the one proposed by Nawaz et al. (1983), which sorts jobs by non-increasing sums of processing times on the machines, fixes the best sequence of the first two jobs, and iteratively inserts all remaining jobs at every possible position of the best sequence kept from the previous iteration.
After obtaining a first schedule, improvement heuristics might be applied to increase the quality of the schedule. Modification rules are applied to search for better solutions until a predefined stopping criteria is met. As scheduling problems' solution spaces are mostly non-convex, it is difficult to avoid being caught in local optima. Improvement can be achieved by limited enumeration techniques or local search approaches.

Metaheuristics

Metaheuristics, in contrast to specifically tailored heuristics, can be categorized as a general algorithmic framework which can be applied to different optimization problems without the need of major modifications. Therefore, these methods need instantiation to solve a given problem.
A metaheuristic is composed of a set of concepts:

- Problem or solution representation,
- initialisation,
- neighbourhood definition,
- neighbourhood search process,

- acceptance criterion,
- termination criterion.

Mostly, metaheuristics also require a initial solution and could be therefore classified as improvement heuristics as well, but they tend to be more robust and flexible and obtain much more refined solutions at the cost of higher computational effort. Due to the non-convexity of combinatorial optimization problems, constructive procedures enhanced by deterministic permanent improvement neighbourhood search approaches easily get stuck in local optima. Metaheuristics therefore systematically expand the search space by allowing for temporal and stochastically worsening of solutions which can be considered for the next neighbourhood search. A large extend of metaheuristics is nature-inspired, i.e. they adapt principles from phenomena of nature to explore a broader variety of rather different solutions. Examples of commonly used search procedures in scheduling problems are simulated annealing, tabu search, genetic algorithms, ant colony algorithms and particle swarm algorithms.

Other Techniques

Some approaches can not be classified as per the categories described above. For example, exact algorithms can be used to derive quick approximate solutions by pre-defining a stopping criteria (e.g. computation time limit). Lagrangian relaxation is another technique of determining an approximate solution with an exact algorithm by applying dualisation of constraints into the objective function. Lagrangian multipliers are used as penalty terms for violations of constraints. Also, hybrid approaches combine exact methods with metaheuristics, e.g. by defining a good initial feasible solution, some post optimisation steps or by decomposing the overall problem into optimally solvable sub-problems and heuristically composing the sub-problems' solutions.
Simulation studies are widely used in dynamic scheduling models to evaluate the performance of different scheduling rules (often basic dispatching rules) under various scenarios. Hybrid approaches arose in this field as well, combin-

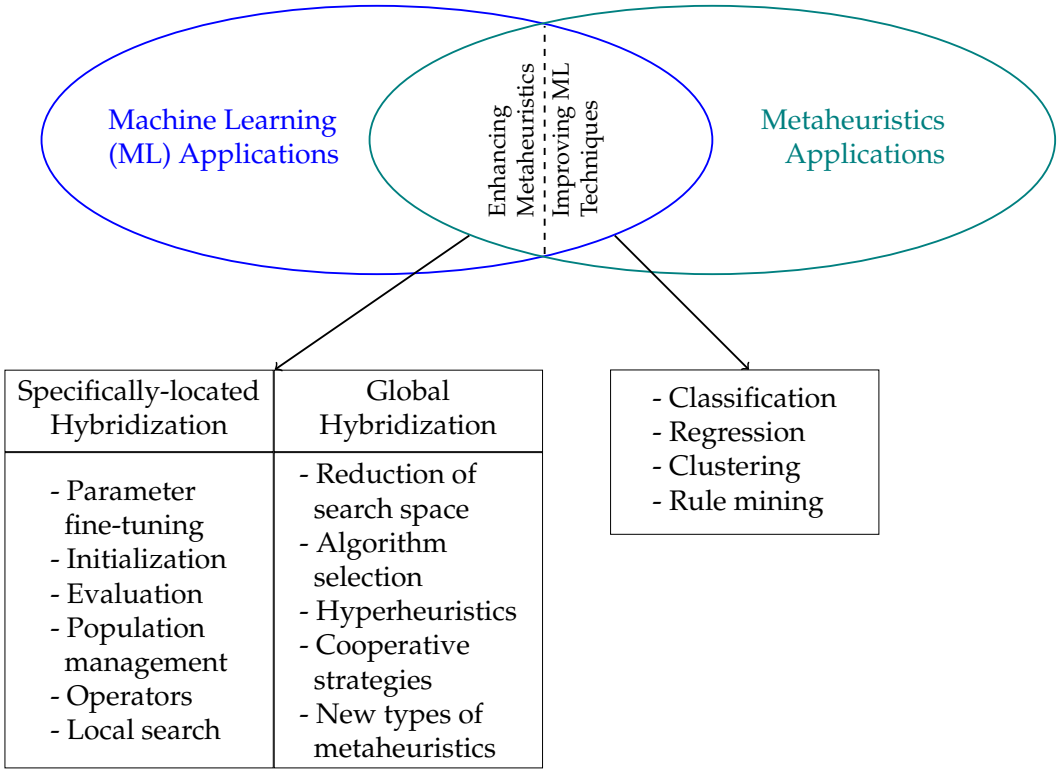


Figure 2.3: Classification of works combining metaheuristics and machine learning. (Calvet et al. (2017)

ing simulation tools and common metaheuristics to solve complex, real-world oriented, problems.

Furthermore, artificial intelligence approaches have been applied to scheduling. The majority of cases follows a hybrid approach of combining machine learning tools and classical scheduling heuristics, e.g. by using machine learning methods to select a best heuristic for a certain current system's state or by replacing some metaheuristic step by a machine learning algorithm. In the area of hyperheuristics, machine learning tools can be used for parameter optimization. Calvet et al. (2017) summarize that the existing literature with regards to the hybridization of metaheuristics and machine learning can be mainly divided into two groups: Approaches where machine learning is employed to enhance metaheuristics, and those in which metaheuristics are used to improve the performance of machine learning techniques. An overview of hybridization strategies is given in figure 2.3 and an overview of machine learning tools will follow in chapter 4.2.

3 The Hybrid Flow Shop Problem

As is apparent from section 3, a broad variety of scheduling problems exists in theory and praxis. This work focusses on the HFS with unrelated machines, as it represents many real-world production settings. Further specifics of the HFS problem are described in the following sections.

3.1 Standard Configuration

In a standard HFS manufacturing system, n jobs have to be processed on m stages with each stage containing a subset M^i $(i = 1, ..., m)$ out of l machines, where $l_i := |M^i|$ equals the number of machines at stage i. Machines arranged at the same stage perform same operations but might differ regarding their speed. A job has to be processed on one of the available machines at every stage consecutively, starting at stage 1. A feasible schedule ensures that no job can be processed by more than one machine at a time and a machine can only process one job at a time. In the standard form, all jobs and machines are available at time zero, machines at a given stage are identical, setup and transportation times are negligible, preemption is not allowed, the capacity of buffers between stages is unlimited and problem data is deterministic. (Ruiz and Vázquez-Rodríguez (2010))

The HFS represents an adequate model for several industrial settings including e.g. the textile industry, cable manufacturing, and electronic manufacturing environment. In these manufacturing systems, the operating procedure follows a flow shop and at some stages machines performing a certain production step are duplicated to achieve additional flexibility or to mitigate the impact of bottleneck stages. (Hidri and Haouari (2011))

Furthermore, from a theoretical perspective the HFS generalizes some fundamental scheduling problems. The special case of each stage including a single machine only transforms the HFS to the much studied flow shop scheduling problem. In the special case where the manufacturing system includes one single stage, the HFS equals the parallel machine problem. Such fundamen-

tal relations are often deployed in decomposition-based solution approaches or while developing upper and lower bounds. (Hidri and Haouari (2011))

HFS problems are, in most cases, NP-complete. Due to their complexity and practical relevance, HFS problems have attracted a lot of attention and a plethora of exact and approximative solution methods have been proposed. (Ruiz and Vázquez-Rodríguez (2010))

An exemplary setting of a typical HFS is shown in figure 3.1:

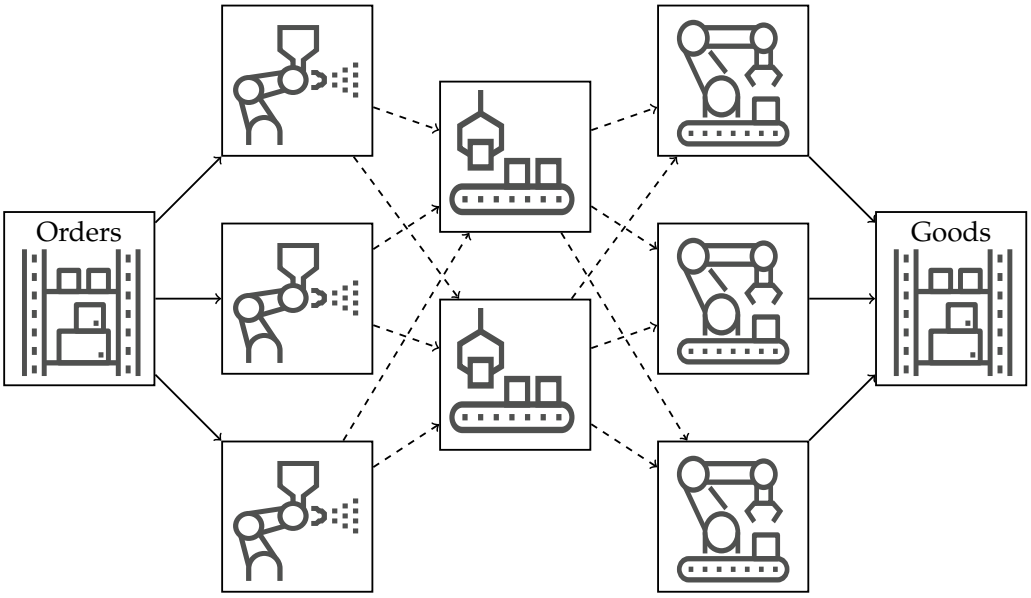


Figure 3.1: The hybrid flow shop system.[5]

[5]Pictures from davooda/Shutterstock.com.

3.1.1 Complexity

In HFS scheduling, two general problems have to be solved simultaneously for all stages, namely the sequencing and machine assignment problem (see figure 3.2). As both sub-problems itself are combinatorial optimization problems, HFS scheduling is a highly complex task.

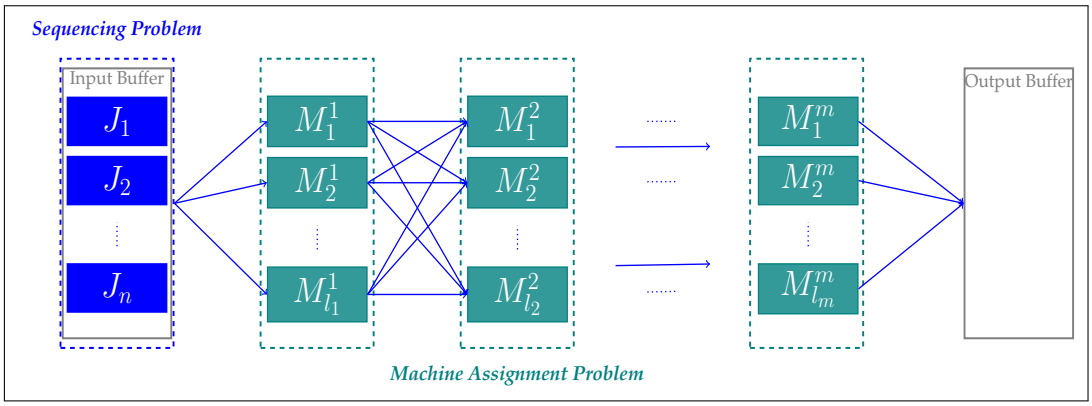


Figure 3.2: The hybrid flow shop sub-problems.

The first sub-problem is to determine a permutation, or sequence, of n jobs on every stage $i \in \{1, ..., m\}$. For each stage, there are $n!$ possible solutions to the sequencing problem. Framinan et al. (2014) provide an illustrative example of the functional dependency of problem size and computation time (see table 3.1). In addition, they show the complexity hierarchy of scheduling problems based on objective function and layout (see figures 3.3 and 3.4). What can be seen is that, even for a simple single machine sequencing problem, the solution space is growing fast with problem size, so that brute force methods are not feasible for real-world problems.

Secondly, for each of the $n!$ permutations of jobs, there exist $(l_i)^n$ possible machine assignments per stage i. As schedules of different stages are mutually dependent, $n!^m \cdot (l_1)^n \cdot (l_2)^n \cdot ... \cdot (l_m)^n$ solutions to the HFS scheduling problem exist. Taking a small example of scheduling 9 jobs in a 3 stage HFS with 4 unrelated parallel machines per stage, the total number of possible schedules is already $9!^3 \cdot 4^9 \cdot 4^9 \cdot 4^9 = 8.6081 \cdot 10^{32}$ which falls under the same category as a single machine sequencing problem with more than 30 jobs, so that it would take at least some billion of years to evaluate all possible solutions according

to table 3.1.

Therefore, choosing an efficient (exact or approximate) algorithm to search for an optimal (or good) solution is crucial and usually arrives at a trade-off between solution quality and computation time.

Table 3.1: Total CPU times for two types of computers for evaluating all solutions in single machine sequencing problems as a function of the number of jobs n. (Framinan et al. (2014))

n	$n!$	CPU Time (Slow*)	Unit	CPU Time (Fast*)	Unit
1	1	1	ns	0.2	fs
2	2	2	ns	0.4	fs
3	6	6	ns	1.2	fs
4	24	24	ns	4.8	fs
5	120	120	ns	24	fs
6	720	720	ns	144	fs
7	5,040	5.04	μs	1.01	ps
8	40,320	40.32	μs	8.06	ps
9	362,880	0.36	ms	72.58	ps
10	3,628,800	3.63	ms	726	ps
11	39,916,800	39.92	ms	7.98	ns
12	479,001,600	479	ms	95.8	ns
13	6,227,020,800	6.23	s	1.25	μs
14	87,178,291,200	87.18	s	17.44	μs
15	1,307,674,368,000	21.79	min	262	μs
16	20,922,789,888,000	349	min	4.18	ms
17	355,687,428,096,000	98.8	h	71.14	ms
18	6,402,373,705,728,000	74.1	days	1.28	s
19	121,645,100,408,832,000	3.85	years	24.33	s
20	2,432,902,008,176,640,000	77.09	years	8.11	min
21	51,090,942,171,709,400,000	16.19	centuries	170.3	min
22	1,124,000,727,777,610,000,000	356	centuries	62.44	h
23	25,852,016,738,885,000,000,000	819	millenia	59.84	days
24	620,448,401,733,239,000,000,000	19.66	million years	3.93	years
25	15,511,210,043,331,000,000,000,000	492	million years	98.3	years
26	403,291,461,126,606,000,000,000,000	12.78	billion years	25.56	centuries
27	10,888,869,450,418,400,000,000,000,000	345	billion years	690.09	centuries
28	304,888,344,611,714,000,000,000,000,000	690	Age of the Universe	1.93	million years
29	8,841,761,993,739,700,000,000,000,000,000	20,013	Age of the Universe	56.04	million years
30	265,252,859,812,191,000,000,000,000,000,000	600,383	Age of the Universe	1.68	billion years

*Slow computer running at 1 GHz, fast computer running at 5 PHz.

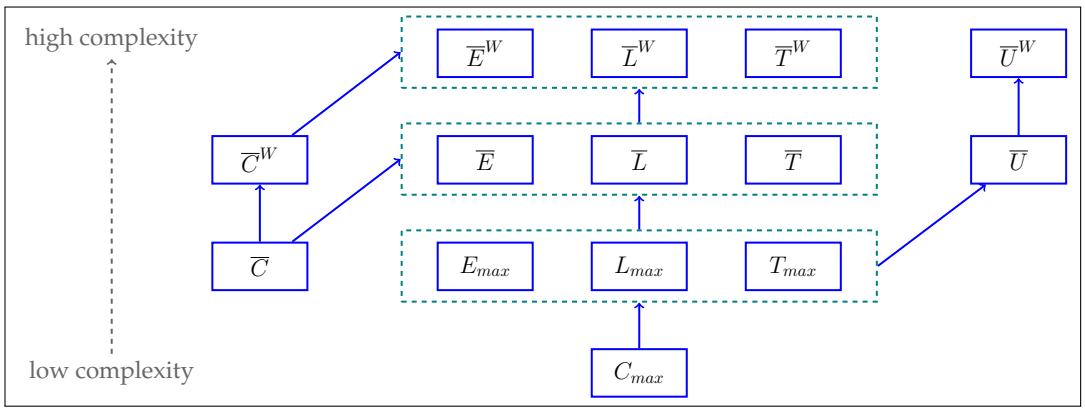


Figure 3.3: Complexity hierarchy with respect to the criteria. (Framinan et al. (2014))

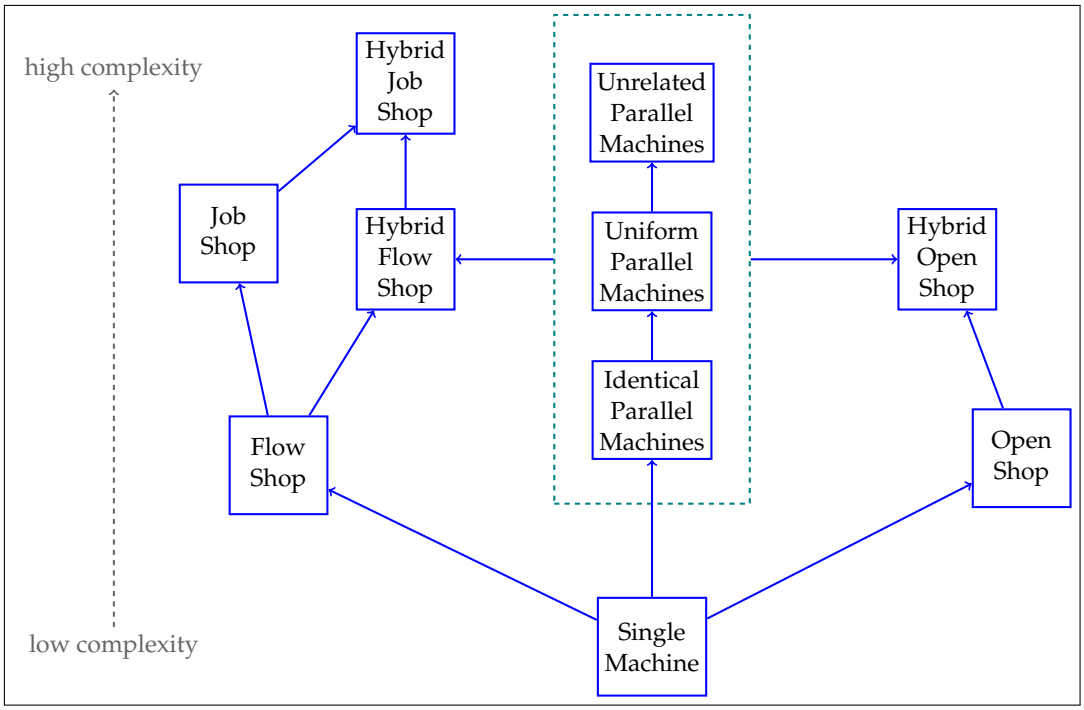


Figure 3.4: Complexity hierarchy with respect to the layouts. (Framinan et al. (2014))

3.1.2 Solution Approaches

An overview of general solution methods for scheduling problems is given in section 3.1.3. Considering HFS problems, all of the above mentioned methods can be applied, whereas, due to their complexity, exact algorithms are usually not suitable.

In practice, dispatching rules are popular as they are suitable to deal with complex, dynamic, and unpredictable environments. From a theoretical point of view, a speciality of HFS scheduling is, that decomposition approaches are often either reducing the HFS to a basic flow shop or to a parallel machine problem, or they are focussing on the bottleneck stage. For such kind of problems, good heuristics are known for most commonly used objective functions, so that it can be built up on those while developing HFS scheduling heuristics. Furthermore, the two sub-problems, sequencing and machine assignment, are often solved separately. (Ruiz and Vázquez-Rodríguez (2010))

A more generic approach is that of instantiation of the HFS to obtain suitable input instances for (mostly nature-inspired) metaheuristics. Although a

plethora of new metaheuristics were proposed during the last 30 years, they usually differ only with respect to the applied strategy for overcoming local optima by adding randomness to deterministic scheduling. Simulated Annealing, Tabu Search and Genetic Algorithms are still the most widely used heuristics in (HFS) scheduling. Furthermore, most metaheuristics proposed for the HFS restrict the search to the sequencing problem and utilize simple priority rules for machine assignment. (Ruiz and Vázquez-Rodríguez (2010))

3.2 State of the Art in Research

For a comprehensive review on HFS literature until 2009 see Ruiz and Vázquez-Rodríguez (2010) and Ribas et al. (2010).[6]

Both reviews state that large scope for future research exists regarding HFS settings with unrelated machines as well as considering other objectives than makespan (see table 3.2 and figure 3.5).

Table 3.2: Percentage of the reviewed papers according to number of stages and type of parallel machines. (Ruiz and Vázquez-Rodríguez (2010))

No. of Stages	**Machine Environment**			**Total**
	Identical	**Uniform**	**Unrelated**	
2	25.12	1.86	4.65	31.63
3	4.19	1.4	0	5.59
m	54.41	1.4	6.97	62.78
Total	**83.72**	**4.66**	**11.62**	**100**

Furthermore, solution approaches are often customized to a specific setting so that a lag of generic methods is identified (see figure 3.6).

An extensive review of HFS literature after the year 2009 is presented, focussing on works considering unrelated machines or machine learning techniques. A summary and classification of reviewed papers is given in table 3.2.

What can be seen is that makespan is still the predominant objective function used in theoretical HFS scheduling and that multi-objective approaches are rather rare. In addition, more and more complex metaheuristics or hybrid

[6] Ruiz and Vázquez-Rodríguez (2010) evaluate more than 200 and Ribas et al. (2010) more than 150 papers.

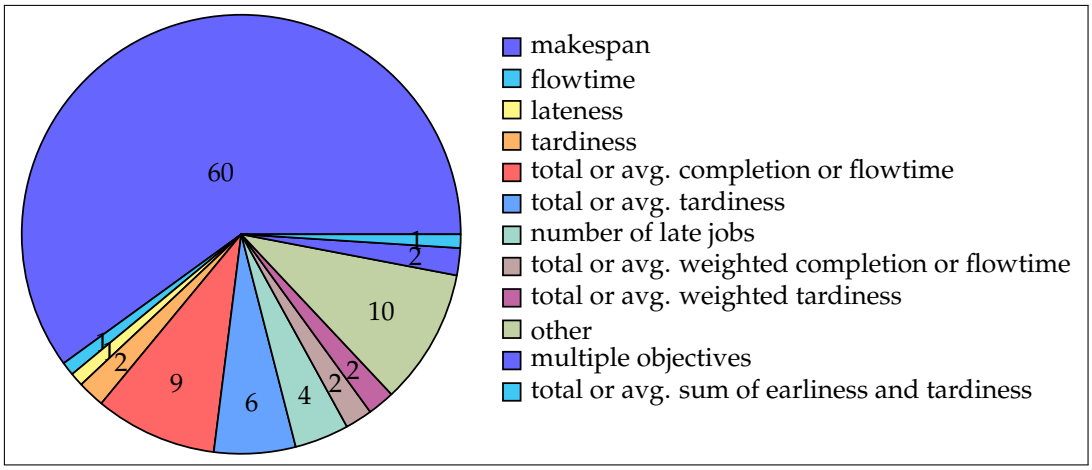


Figure 3.5: Distribution of objective functions (in %). (Ruiz and Vázquez-Rodríguez (2010))

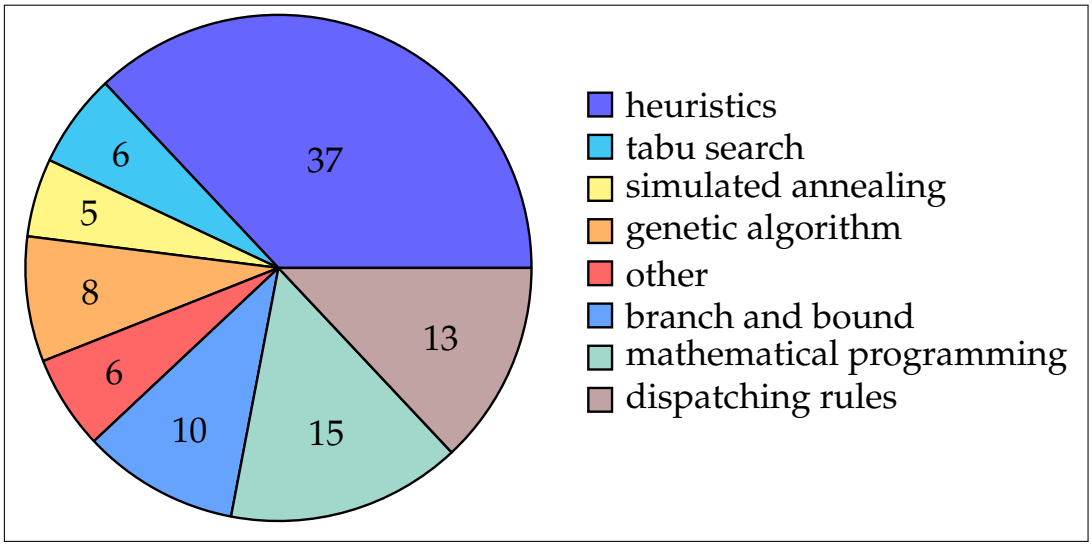


Figure 3.6: Distribution of employed techniques (in %). (Ruiz and Vázquez-Rodríguez (2010))

approaches combining different nature-inspired search strategies are proposed. While an increase in work considering unrelated machines can be detected, many authors still focus on simple 2-stage HFS and/or identical machines. To the best of our knowledge, no author has considered machine learning techniques in HFS scheduling yet to replace one step of a heuristic. An overview of approaches including ML algorithms in other scheduling-related problems is provided in section 4.

The m-stage HFS with Unrelated Machines

All papers presented chronologically in this section are published after the year 2009 and consider a variant of the $\mathrm{FH}_m, ((\mathrm{RM}^k)_{k=1}^m) \mid \cdot \mid \cdot)$ problem.
Sequence-dependent setup times and blocking are included in a HFS problem to minimize the makespan and maximum tardiness criteria by Rashidi et al. (2010). An algorithm based on independent parallel GAs by dividing the whole population into multiple sub-populations. Each sub-population is assigned with different weights to search for optimal solutions in different directions. To find Pareto solutions, each algorithm is combined with a local search step and a newly proposed redirect procedure which tries to help the algorithm to overcome the local optima. A method to speed up the searching procedure is presented. If the local search step failed to work, then the redirect procedure changes the direction and refreshes the population.
Urlings et al. (2010) introduce advanced GAs for a complex HFS problem with a makespan objective. General precedence constraints among jobs, machine release dates, time lags and sequence dependent (anticipatory and non-anticipatory) setup times are taken into account. The model represents a ceramic tile production system. The introduced algorithms employ solution representation schemes with different degrees of directness and new machine assignment rules are tested. The different GAs are compared among each other and to some benchmark heuristics, indicating that simple solution representation schemes result in the best performance and that the GAs outperform other heuristics.
Zandieh et al. (2010) try to narrow the gap between real-world industries scheduling and the production scheduling theories by considering relevant constraints, i.e. sequence-dependent setup times, machine release date and time lags. A GA is proposed and response surface methodology is applied to set the parameters of GA and to estimate the proper values of GA parameters in continually intervals. The solutions of GA are compared with existing heuristic from literature such as SPT, LPT and NEH.
Lot streaming is the technique of splitting a given job into sublots to allow the

overlapping of successive operations. Defersha (2011) and Defersha and Chen (2012) include lot streaming and stage skipping in their MILP representation and propose a GA, which is executed on both sequential and parallel computing platforms. Numerical examples showed that the parallel implementation greatly improved the computational performance of the developed heuristic.
Limited waiting times between every two successive processing operations and ready time of jobs are studied by Attar et al. (2013). A metaheuristic named biogeography-based optimization is proposed and compared to imperialist competitive algorithm and population-based SA. Response surface methodology is used for parameter tuning.
Lately, explicitly energy-related objective functions are considered in scheduling literature, although energy consumption is somewhat dependent on machine utilization and job completion times. However, classical objectives do not take energy consumption or environmental impacts completely into account.Therefore, Dai et al. (2013) propose an energy-efficient HFS model. An MILP based on an energy-efficient mechanism is presented considering makespan and energy consumption a minimization criteria. multi-objective optimization. A hybrid bi-objective metaheuristic, combining GA and SA is proposed and a case study of a metal working workshop is conducted which shows, that makespan and energy consumption may be conflicting objectives.
A robotic scheduling problem in blocking HFS cells that consider multiple part types, multiple robots and machine eligibility constraints is evaluated by Elmi and Topaloglu (2013). A MILP formulation with makespan objective is given and Simulated Annealing (SA) based solution approach with a new neighborhood structure based on blocking conditions is developed. They conclude that the appropriate number of robots depends on the sequence of processing operations to be performed at each stage.
Li and Pan (2015) propose a novel hybrid algorithm that combines the artificial bee colony and tabu search to solve the HFS scheduling problem with limited buffers with makespan objective. A novel decoding method for limited buffer constraints is utilized and four neighborhood structures are embedded

to balance the exploitation and exploration abilities of the algorithm and tabu search is used to achieve a learning ability for producing neighboring solutions in different promising regions and to enhance the search ability of the employed bees and onlookers. Parameter setting is investigated by using the Taguchi method of design of experiment.

Soltani and Karimi (2015) emphasis that just-in-time philosophy (which tries to fulfill each customer's order in exact size of demand and at the earliest time which is preferred) led to a change of production policy from batch production of a single type to batch production mixed of different types, called minimal part set. With this motivation, cyclic policy which attempts to reduce work-in-process inventory arouse. A MILP is presented, including eligibility and limited buffer constraints. Different heuristics and metaheuristics are proposed and compared, with SA -using some embedded heuristics- is performing best.

A scheduling problem that occurs in potash mining is introduced by Schulze et al. (2016), where a block excavation sequence has to be found taking into account a limited number of underground machines as well as safety-related restrictions. The aim is to minimize the makespan (here: maximum completion time of excavations). Reentry and job-precedences are included in the HFS model formulation and a MILP model is presented. A multistart heuristic is developed based on a specific priority rules and a modified version of the Giffler and Thompson procedure is applied.

In two publications, Shahvari and Logendran (2016) and Shahvari and Logendran (2018) address the HFS batch scheduling problem with sequence- and machine-dependent family setup times where the objective is to simultaneously minimize the weighted sum of the total weighted completion time and total weighted tardiness. In order to reflect the industry requirements, machine availability times, job release times, machine capability and eligibility for processing jobs, stage skipping, and learning effect are considered. Here, batch scheduling disregards the group technology assumptions by splitting predetermined groups of jobs into inconsistent batches. The authors integrate the

batching decision into the group scheduling approach. Metaheuristics based on hybridization of local search and population-based structures along with stage-based interdependency strategy are proposed and an initial solution finding mechanism and a comprehensive data generation mechanism are developed. Lower bounds are obtained by two MILP models to evaluate the algortithms' performance.

Zabihzadeh and Rezaeian (2016) consider a HFS with machine eligibility constraints and job release dates. In addition, blocking restriction is considered as robots perform transportation, loading and unloading of parts. The objective is to find an optimal sequence of processing parts and robots movements to minimize the makespan and finding the closest number to the optimal number of robots. A MILP model for the problem which considers release dates and transportation times is presented. Ant colony optimization (ACO) with double pheromone and a GA are proposed. A comparison shows that the GA performs better than ACO and the near optimal numbers of robots are determined.

A HFS problem with stage skipping is solved by de Siqueira et al. (2018) through a multiobjective variable neighborhood search metaheuristic, the two evaluation criteria being the minimization of the makespan and the minimization of the weighted sum of tardiness. Instances from literature are solved by four versions of the proposed algorithm and the results are evaluated using the hypervolume, epsilon, spacing and sphere counting metrics.

A real-world environment of the tortilla industry is presented by Yaurima-Basaldua et al. (2018). The HFS model considers multiproduct and batch production while optimizing completion time and energy consumption criteria. The proposed bi-objective algorithm is based on the known nondominated sorting genetic algorithm II. Statistical analysis of multifactorial variance is deployed and a branch and bound algorithm is used to assert obtained performance. Using real data from a tortilla producer, 48% of production time and 47% of electricity consumption can be saved.

Yu et al. (2018) present a GA to solve the HFS problem to minimize total

tardiness. Practical constraints as unrelated machines and machine eligibility are considered. A new decoding method is developed, which is able to obtain tight schedule and guarantees the influence of the chromosome on the schedule. A full factorial design of experiment for parameter calibration is done and the proposed GA is compared to several calibrated state-of-art algorithms on 450 instances with different size and correlation patterns of operation processing time.

From figure 3.7 it becomes apparent that researchers have recognized the importance of the unrelated machine case[7] but that makespan remains the predominant objective function. Metaheuristics are by far the most widely applied solution method, and more and more complex bio-inspired search strategies are proposed. ML techniques are rarely considered in HFS literature, although the combination with metaheuristics in hybrid approaches produces promising results in other areas of (scheduling) research.

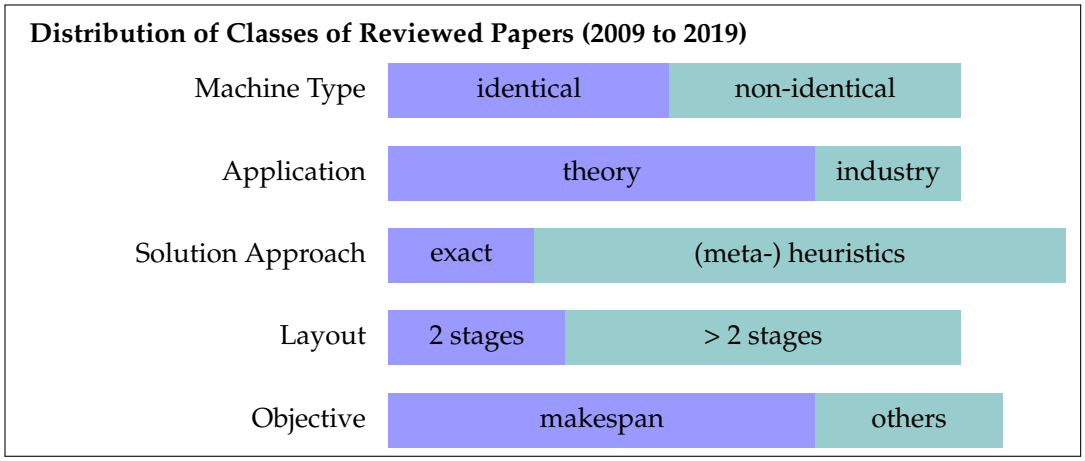


Figure 3.7: Overview of reviewed papers from the years 2010 to 2019 grouped by different classes.

[7]The presented papers focus on unrelated machines and therefore literature with identical machines has only been included if there was some other relevance for this work. Therefore, the presented ratio is biased towards unrelated machine cases.

Table 3.3: Recent publications in the field of HFS scheduling.

Author and Year	Problem	Solution Method
Boudhar and Meziani (2010)	$\mathrm{FH}_2, (1^1, \mathrm{P2}^2 \mid \mathrm{rcrc} \mid C_{\max})$	heuristic based on dispatching rules
Figielska (2010)	$\mathrm{FH}_2, ((\mathrm{RM}^k)_{k=1}^2) \mid \mathrm{prmp}, \mathrm{rc} \mid C_{\max})$	heuristics based on linear programming
Luo et al. (2012)	$\mathrm{FH}_4, ((\mathrm{PM}^k)_{k=1}^4) \mid S_{sd} \mid C_{\max})$	genetic algorithm, heuristic based on decoding scheme of proposed genetic algorithm
Mouelhi-Chibani and Pierreval (2010)	$\mathrm{FH}_2, ((\mathrm{P2}^k)_{k=1}^2) \mid \mathrm{stoch}, \mathrm{unavail}, \mathrm{rcrc} \mid \overline{T})$	simulation, neural network for dispatching rule selection, weight vector provided by simulated annealing
Naderi et al. (2010)	$\mathrm{FH}_m, ((\mathrm{PM}^k)_{k=1}^m) \mid S_{sd}, \mathrm{skip} \mid C_{\max})$	heuristic based on dispatching rules, iterated local search
Rashidi et al. (2010)	$\mathrm{FH}_m, ((\mathrm{RM}^k)_{k=1}^m) \mid \mathrm{block}, S_{sd} \mid C_{\max}, T_{max})$	multi-objective genetic algorithm with redirect procedure
Scholz-Reiter et al. (2010)	$\mathrm{FH}_m, ((\mathrm{UM}^k)_{k=1}^m) \mid S_{sd} \mid C_{\max})$	autonomous control
Urlings et al. (2010)	$\mathrm{FH}_m, ((\mathrm{RM}^k)_{k=1}^m) \mid M_j, \mathrm{prec}, \mathrm{lag}, S_{sd}, r_j \mid C_{\max})$	genetic algorithms
Zandieh et al. (2010)	$\mathrm{FH}_m, ((\mathrm{RM}^k)_{k=1}^m) \mid M_j, S_{sd}, \mathrm{lag} \mid C_{\max})$	genetic algorithm
Choi et al. (2011)	$\mathrm{FH}_5, (\mathrm{P10}^1, \mathrm{P20}^2, \mathrm{P10}^3, \mathrm{P15}^4, \mathrm{P10}^5) \mid \mathrm{stoch}, \mathrm{rcrc}, \mathrm{unavail}, \mathrm{block} \mid \overline{F}, \overline{T}, \overline{U}, \mathrm{throughput})$	simulation, decision tree for dispatching rule selection
Defersha (2011)	$\mathrm{FH}_m, ((\mathrm{RM}^k)_{k=1}^m) \mid \mathrm{lag}, S_{sd}, M_j \mid C_{\max})$	MILP
Hekmatfar et al. (2011)	$\mathrm{FH}_2, ((\mathrm{PM}^k)_{k=1}^2) \mid S_{sd}, \mathrm{rcrc} \mid C_{\max})$	modified SPT, LPT, Johnson and NEH Algorithm, hybrid genetic algorithm
Defersha and Chen (2012)	$\mathrm{FH}_m, ((\mathrm{RM}^k)_{k=1}^m) \mid \mathrm{lag}, S_{sd}, M_j \mid C_{\max})$	MILP and genetic algorithm
Gicquel et al. (2012)	$\mathrm{FH}_4, ((\mathrm{PM}^k)_{k=1}^4) \mid \mathrm{block}, \mathrm{no\text{-}wait}, \mathrm{size}_{j3}, S_{si} \mid \overline{T}^W)$	MILP, cut & branch algorithm
Liao et al. (2012)	$\mathrm{FH}_m, ((\mathrm{PM}^k)_{k=1}^m) \mid\mid C_{\max})$	hybrid particle swarm optimization incl. bottelneck heuristic and simulated annealing
Niu et al. (2012)	$\mathrm{FH}_m, ((\mathrm{PM}^k)_{k=1}^m) \mid\mid \overline{F})$	quantum immune algorithm

Author and Year	Problem	Solution Method
Trabelsi et al. (2012)	$\mathrm{FH}_m, ((\mathrm{PM}^k)_{k=1}^m) \mid \mathsf{block} \mid C_{\max}$	MILP, lower bound
Yao et al. (2012)	$\mathrm{FH}_2, ((\mathrm{PM}^k)_{k=1}^2) \mid \mathsf{batch}, \mathsf{lim\text{-}wait} \mid C_{\max}$	dynamic programming for problems of class P, heuristics for problems of class NP
Almeder and Hartl (2013)	$\mathrm{FH}_2, (1^1, (\mathrm{R2}^2)_{k=1}^2) \mid \mathsf{batch},\ \mathsf{stoch}, \mathsf{prmu} \mid \mathsf{utilization})$	variable neighborhood search
Attar et al. (2013)	$\mathrm{FH}_m, ((\mathrm{RM}^k)_{k=1}^m) \mid \mathsf{lim\text{-}wait}, r_j \mid C_{\max}$	MILP, biogeography-based optimization: Imperialist competitive and simulated annealing
Dai et al. (2013)	$\mathrm{FH}_m, ((\mathrm{RM}^k)_{k=1}^m) \mid\mid C_{\max}, \mathsf{energy\ consumption})$	MILP, genetic simulated annealing algorithm
Elmi and Topaloglu (2013)	$\mathrm{FH}_m, ((\mathrm{RM}^k)_{k=1}^m) \mid \mathsf{block}, M_j \mid C_{\max}$	MILP, simulated annealing
Li et al. (2013)	$\mathrm{FH}_2, ((\mathrm{RM}^k)_{k=1}^2) \mid \mathsf{batch}, r_j \mid \overline{T})$	combination of dispatching rules
Rossi et al. (2013)	$\mathrm{FH}_2, ((\mathrm{PM}^k)_{k=1}^2) \mid \mathsf{batch}, r_j, S_{si} \mid C_{\max}, \overline{U})$	MILP, heuristic
Wang and Liu (2013a)	$\mathrm{FH}_2, (1^1(\mathrm{PM}^k)_{k=1}^2) \mid \mathsf{no\text{-}wait} \mid C_{\max}$	genetic algorithm and simple heuristics
Wang and Liu (2013b)	$\mathrm{FH}_2, (1^1, (\mathrm{PM}^k)_{k=1}^m) \mid M_j \mid C_{\max}$	heuristics based on branch & bound
Fattahi et al. (2014)	$\mathrm{FH}_2, ((\mathrm{PM}^k)_{k=1}^2) \mid S_{si} \mid C_{\max}$	branch & bound
Figielska (2014)	$\mathrm{FH}_2, (1^1, \mathrm{RM}^2 \mid \mathsf{prmp},\ \mathsf{rc} \mid C_{\max}$	dispatching rules, column generation algorithm (incl. linear programming, tabu search, greedy procedure)
Li et al. (2014)	$\mathrm{FH}_m, ((\mathrm{PM}^k)_{k=1}^m) \mid\mid C_{\max}$	hybrid variable neighborhood search incl. chemical-reaction optimization and estimation of distribution
Marichelvam et al. (2014)	$\mathrm{FH}_m, ((\mathrm{PM}^k)_{k=1}^m) \mid\mid C_{\max}$	improved cuckoo search incl. NEH
Naderi et al. (2014)	$\mathrm{FH}_m, ((\mathrm{PM}^k)_{k=1}^m) \mid\mid C_{\max}$	MILP, hybrid particle swarm optimization algorithm incl. local search and tabu search,
Pan et al. (2014)	$\mathrm{FH}_m, ((\mathrm{PM}^k)_{k=1}^m) \mid\mid C_{\max}$	artificial bee colony algorithm, 24 basic heuristics
Rabiee et al. (2014)	$\mathrm{FH}_2, ((\mathrm{RM}^k)_{k=1}^2) \mid \mathsf{no\text{-}wait}, S_{sd}, r_j, \mathsf{rcrc} \mid C_{\max}$	hybrid algorithm: Imperialist competetive alogrithm, simulated annealing, variable neighborhood search, genetic algorithm

Author and Year	Problem	Solution Method
Cui and Gu (2015)	$\mathrm{FH}_m, ((\mathrm{PM}^k)_{k=1}^m) \mid\mid C_{\max}$	MILP, improved discrete artificial bee colony algorithm
Jiang et al. (2015)	$\mathrm{FH}_m, ((\mathrm{PM}^k)_{k=1}^m) \mid S_{si}$, batch, controllable processing times $\mid$ $\overline{T}, \overline{E}$, waiting time, adjusting cost)	bi-layer decomposition approach: Hybrid differential evolution algorithm incl. variable neighborhood search for last stage and list scheduling for other stages
Jun and Park (2015)	$\mathrm{FH}_2, ((\mathrm{RM}^k)_{k=1}^2) \mid r_j \mid \overline{T}$)	NEH, local search, dispatching rule
Li et al. (2015)	$\mathrm{FH}_3, ((\mathrm{RM}^k)_{k=1}^3) \mid$ batch $\mid C_{\max}$)	ten different heuristics
Li and Pan (2015)	$\mathrm{FH}_m, ((\mathrm{RM}^k)_{k=1}^m) \mid$ block $\mid C_{\max}$)	hybrid artificial bee colony algorithm incl. tabu search
Nikzad et al. (2015)	$\mathrm{FH}_2, (\mathrm{UM}^1, \mathrm{R2}^2 \mid M_j, S_{sd} \mid C_{\max})$	MILP, hybrid imperialist competitive algorithm incl. simulated annealing
Sangsawang et al. (2015)	$\mathrm{FH}_2, (1^1, \mathrm{PM}^2 \mid$ block, rcrc $\mid C_{\max})$	hybrid genetic algorithm, hybrid particle swarm optimization
Soltani and Karimi (2015)	$\mathrm{FH}_m, ((\mathrm{RM}^k)_{k=1}^m) \mid$ batch, block, $M_j \mid \overline{F})$	MILP, greedy assignment algorithm, genetic algorithm, simulated annealing
Lin et al. (2016)	$\mathrm{FH}_m, ((\mathrm{PM}^k)_{k=1}^m) \mid$ stoch $\mid C_{\max})$	branch & bound algorithm
Rahmani and Ramezanian (2016)	$\mathrm{FH}_m, ((\mathrm{PM}^k)_{k=1}^m) \mid$ stoch $\mid \overline{T}^w$, stability)	variable neighborhood search and reactive rescheduling approach
Schulze et al. (2016)	$\mathrm{FH}_m, ((\mathrm{RM}^k)_{k=1}^m) \mid$ rcrc, prec $\mid C_{\max})$	MILP, multistart heuristic based on dispatching rules
Shahvari and Logendran (2016)	$\mathrm{FH}_m, ((\mathrm{RM}^k)_{k=1}^m) \mid$ batch, S_{sd}, M_j, skip $\mid \overline{T}^w, \overline{C}^w)$	MILP, metaheuristics based on tabu search incl. path-relinking
Zabihzadeh and Rezaeian (2016)	$\mathrm{FH}_m, ((\mathrm{RM}^k)_{k=1}^m) \mid$ block, M_j, r_j, rc $\mid C_{\max})$	MILP, genetic algorithm, ant colony optimization
Lalitha et al. (2017)	$\mathrm{FH}_m, (1^k)_{k=1}^{m-1}, \mathrm{PM}^m \mid$ batch $\mid C_{\max})$	MILP, heuristic algorithm
Pan et al. (2017a)	$\mathrm{FH}_m, ((\mathrm{PM}^k)_{k=1}^m) \mid S_{sd} \mid C_{\max})$	6 local search heuristics, fruit fly optimization, migrating birds optimization, artificial bee colony optimization

Author and Year	Problem	Solution Method
Pan et al. (2017b)	$\mathrm{FH}_m, ((\mathrm{PM}^k)_{k=1}^m) \mid\mid \overline{T}^w, \overline{E}^w)$	heuristics based on iterated greedy and iterated local search
de Siqueira et al. (2018)	$\mathrm{FH}_m, ((\mathrm{RM}^k)_{k=1}^m) \mid M_j, \mathsf{skip} \mid C_{\max}, \overline{T}^w)$	multi-objective variable neighborhood search
Dios et al. (2018)	$\mathrm{FH}_m, ((\mathrm{PM}^k)_{k=1}^m) \mid \mathsf{skip} \mid C_{\max})$	skipping property-related heuristics
Fernandez-Viagas et al. (2018)	$\mathrm{FH}_m, ((\mathrm{PM}^k)_{k=1}^m) \mid\mid C_{\max})$	comparison of 24 constructive heuristics
Shahvari and Logendran (2018)	$\mathrm{FH}_m, ((\mathrm{RM}^k)_{k=1}^m) \mid \mathsf{batch}, S_{sd}, M_j, \mathsf{skip}, \mathsf{learning} \mid \overline{T}^w, \overline{C}^w)$	MILP, metaheuristics based on tabu search incl. path-relinking, particle swarm optimization and local search
Yu et al. (2018)	$\mathrm{FH}_m, ((\mathrm{RM}^k)_{k=1}^m) \mid M_j \mid \overline{T})$	genetic algorithm
Yaurima-Basaldua et al. (2018)	$\mathrm{FH}_6, (\mathrm{P6}^1, \mathrm{R6}^2, \mathrm{P6}^3, (\mathrm{R6}^k)_{k=4}^6 \mid S_{sd}, S_{si}, \mathsf{batch}, \mathsf{block} \mid C_{\max}, \mathsf{energy\ consumption})$	multi-objective nondominated sorting genetic algorithm II, branch & bound

4 Machine Learning in the Context of Scheduling

A wide range of research deals with the application and optimization of machine learning (ML) models in a variety of areas from engineering, business analytics, to health care applications, and others. This work focusses on ML applications in scheduling, so that only information relevant in this context is presented in the following sections.

4.1 Basic Concept of Machine Learning

Machine learning algorithms, as a subset of artificial intelligence in the field of computer science, are aiming at performing tasks by generalizing from examples without being manually programmed. With progressive digitization, as more data becomes available, more ambitious problems can be tackled. (Domingos (2012))

ML can also be defined as programming computers to optimize a performance criterion based on example data or past experience. A computer program is executed to learn optimal parameters of the (predictive or descriptive) model, which has only been defined up to some parameters in advance, using training data or past experience. Building ML models involves the theory of statistics as the core task is to draw inference from a sample, or to generalize from samples. Generalization, or knowledge extraction, is achieved by induction, i.e. finding simple models that explain the observed data by extracting general rules from a set of particular cases. (Alpaydin (2009))

According to Witten et al. (2016), generalization is the main task of ML and leads to a concept description as a result of learning, where learning is defined as the search through an enormous but finite search space of possible rules (or concepts) to predict output based on data. This means that based on given data, an algorithm for prediction of the desired output is learned. Learning strategies can be either based on enumerating concepts and striking out those that do not perform well or on hill climbing approaches on the search space to iteratively find the rules that perform best for the given data with respect to

a pre-specified performance measure.
Building and training ML models includes several heuristic steps. Thus, accuracy levels of less than 100 % are common and the aim is to find a concept description that leads to an acceptable error rate.
Bias, in general, can be defined as a learner's tendency to consistently learn the same wrong thing whereas variance is the tendency to learn random things irrespective of the real signal. Bias and variance are related to a model's complexity. Bias indicates that the model does not contain the optimal solution (= underfitting, low complexity) whereas variance appears if the model also learns the noise inherent in the training data (= overfitting, high complexity). This conflict is also known as the bias/variance dilemma and a variety of model selection procedures take this trade-off into account to select an optimal complexity level. (Alpaydin (2009))
A concretisation of the basic choices in ML are e.g. the dimensions of supervised ML algorithms:

- Model selection (hypothesis class and parameters),
- Loss function (definition of approximation error),
- Optimization procedure to find optimal parameters (selection of best hypothesis).

Although it is often discussed, no real dividing line can be drawn between statistics and ML as both research streams can be seen as a continuum of data analysis techniques, and overlap to a large extent and very similar schemes have developed in parallel in ML and statistics. (Witten et al. (2016))
Choosing the right problem representation, learning algorithm and training strategy is a complicated task and there are many components to be considered as can be seen in table 4.1.
Nevertheless, Data Engineering in form of pre- and post-processing of data is of equal importance. Problems arising in this area are, for example, considering optimal strategies for attribute selection, attribute discretization, data projections, sampling, and data cleansing. (Domingos (2012))

Table 4.1: Components of learning algorithms. (Domingos (2012))

Representation	**Evaluation**	**Optimization**
Instances	Accuracy/Error Rate	Computational Optimization
K-nearest neighbor	Precision and Recall	Greedy Search
Support Vector Machines	Squared Error Likelihood	Beam Search
Hyperplanes	Posterior Probability	Branch-and-Bound
Naive Bayes	Information Gain	Continuous Optimization
Logistic Regression	K-L Divergence	Unconstrained
Decision Trees	Cost/Utility	Gradient Descend
Sets of Rules	Margin	Conjugate Gradient
Propositional Rule		Quasi-Newton Methods
Logic Programs		Constrained
Neural Networks		Linear Programming
Graphical Models		Quadratic Programming
Bayesian Networks		
Conditional Random Fields		

In this work, as synthetic data is used, the main decision is to select suitable ML algorithms out of the many algorithms available. Therefore, a preliminary study was conducted, where neural networks and support vector machines proved to be suitable for the given task and thus will be considered in the following.

4.2 Machine Learning Algorithms

This work applies ML tools to enhance a newly proposed job scheduling heuristic, employing existing models implemented in the software *EIDOminer*. Therefore, only basics regarding the broad field of ML algorithms are presented.

Supervised and Unsupervised Learning

Supervised learning targets at finding a mapping from the input pattern to an output whose correct values are provided by a supervisor. (Alpaydin (2009)) It refers to working with a set of labelled training data, meaning that for every input pattern the corresponding output value is known. In unsupervised learning, no supervisor exists and training is not performed based on pre-labelled data. Instead, the algorithm autonomously finds a hidden pattern in large amounts of data. (Bell (2014))

Only input data is presented to such ML algorithms and the aim is to find regularities in the input, i.e. a structure in the input space such that certain patterns occur more often than others. (Alpaydin (2009))
In this way, there is no clear result in terms of "right" or "wrong" answers; instead, unsupervised ML algorithms are run to explore what patterns and outcomes occur. Therefore, unsupervised learning can be seen more as a case of data mining rather than of actual learning and is predominantly found in clustering tasks. (Bell (2014))
Semi-supervised learning algorithms are trained on a combination of labelled and unlabelled data, typically the fraction of labelled data being much smaller than amount of unlabelled data. The aim is to considerably improve the learning accuracy of unsupervised learning (where no data is labelled), but without the time and costs required for supervised learning (where all data needs to be labelled). (Chapelle et al. (2009))
Reinforcement learning is used in applications where the output of a system is a sequence of actions, single actions being less important than the *policy*, i.e. the sequence of correct actions to reach the objective. The goodness of actions in intermediate states is assessed and learned from past good action sequences by the ML algorithm to be able to generate a policy. (Alpaydin (2009))
In this work, only supervised learning is applied.

Classification and Regression

Classification aims at assigning an unknown pattern (i.e. an instance of input data) to one out of a finite number of classes that are considered to be known. Initially, any ML task starts with the decision of how to formally represent each pattern. During the preprocessing stage, related information (that resides in the raw data) has to be encoded in an efficient and information-rich way. Usually, the raw data is transformed in a new space with each pattern represented by a vector $x \in \mathbb{R}^l$, the feature vector, its l elements being the features, and thus becomes a single point in an l-dimensional space, known as the feature or input space. (Theodoridis (2015))

The classifier is trained in a next step. For example, in supervised learning, a set of data with known classes is defined as the training set. This is a set of pairs, (y_n, x_n), $n = 1, 2, ..., N$, where y_n is the class label or (output) variable denoting the class to which x_n belongs. Class labels take values over a discrete set, $\{1, 2, ..., M\}$, for an M-class classification task. Based on the training data, a function f is designed which predicts the output label given the input, i.e. measured values of the features. This function is known as the *classifier*. In general, a set of such functions has to be selected and there exist many algorithms to determine, train and test suitable prediction functions. Afterwards, given an unknown pattern, the corresponding feature vector x determined from the raw data is presented to the classifier and, according to the value $f(x)$, the pattern is classified in one of the M classes. (Theodoridis (2015))

To a large extent, regression shares the feature generation/selection stage of classification. The main difference is that the output variable y is not discrete but takes values in an interval on the real axis or in a region on the complex plane. Therefore, a regression task is basically comparable to a curve fitting problem. For a given set of training points, (y_n, x_n), $n = 1, 2, ..., N$, $y_n \in \mathbb{R}$, $x_n \in \mathbb{R}^l$, a function f is to be designed whose graph fits the data. Again, a variety of algorithms exist for finding, training and testing such a prediction function which is then able to predict output values for unknown input patterns. (Theodoridis (2015))

Since ML is employed to predict an objective function value (real number) in one step of our heuristic, this work will focus on regression problems in the following.

4.3 Specific Methods of Supervised Learning

The ML algorithms that are employed in the proposed prediction model will be briefly described in the following sections. In addition, an overview on fusion techniques is presented.

Support Vector Machines

The support vector machine (SVM) is a supervised learning algorithm suitable for classification and regression problems. The main idea of this method is to use a hyperplane to separate data. If such data is not linearly separable, a *kernel*[8] is used to transform data into a higher dimension to achieve separability. The optimal hyperplane, among all separating hyperplanes, is distinguished by the maximum margin of separation between any training point and the hyperplane, and determined by solving a corresponding optimization problem. The positioning of the hyperplane is completely described by the coordinates of *support vectors*, which lie on the margin and are the data points most difficult to classify. All other training examples being irrelevant, this subset of the training instances represents the learned knowledge of the model. (Scholkopf and Smola (2001))

Neural Networks

The layered structure of a neural network[9] (NN) resembles the networked structure of *neurons* in the brain, with *layers* of connected *nodes*. NN can perform supervised and unsupervised learning and can generate prediction functions for classification and regression.

Input data is broken down into layers of abstraction. It can be trained over many examples to recognize patterns in a similar way as the human brain does. A NN model is defined by the connections of its individual nodes (or neurons) and by the strength, or weights, of those connections. These weights are automatically adjusted during training according to a specified learning rule until a sufficient prediction quality is achieved. In this work, we only consider backpropagation NN (BPN), where neuron weights are updated based

[8] Kernel functions enable algorithms to operate in a high-dimensional space by utilizing dot products; that is, via a function evaluation performed in the original low-dimensional space. This property is also known as the kernel trick. The kernel trick avoids the explicit mapping that is needed to get linear learning algorithms to learn a nonlinear function or decision boundary. (Theodoridis (2015)) A detailed description of kernel learning methods is provided by Scholkopf and Smola (2001).

[9] The definition here includes only NN without recurrence, i.e. feedforward NN. For recurrent artificial NN, e.g. the Hopfield NN, see Hopfield (1982).

on the gradient of the loss function with respect to each weight by the chain rule, iterating backwards one layer at a time from the last layer, i.e. the error propagates from the output back to the inputs. (Alpaydin (2009))
Inspired by biological nervous systems, NN combine several processing layers, using simple elements operating in parallel, consisting of an input layer, one or more hidden layers, and an output layer. The layers are interconnected via nodes, or neurons, with each layer using the output of the previous layer as its input. The output of a single node thereby depends on the input values, the respective weights, the bias value (critical threshold for activation) and the chosen activation function. (Bell (2014))

Fusion Methods for Algorithms

In practice, a variety of ML algorithms exist and an overview of different ML algorithms by previously described categories is given in figure 4.1.

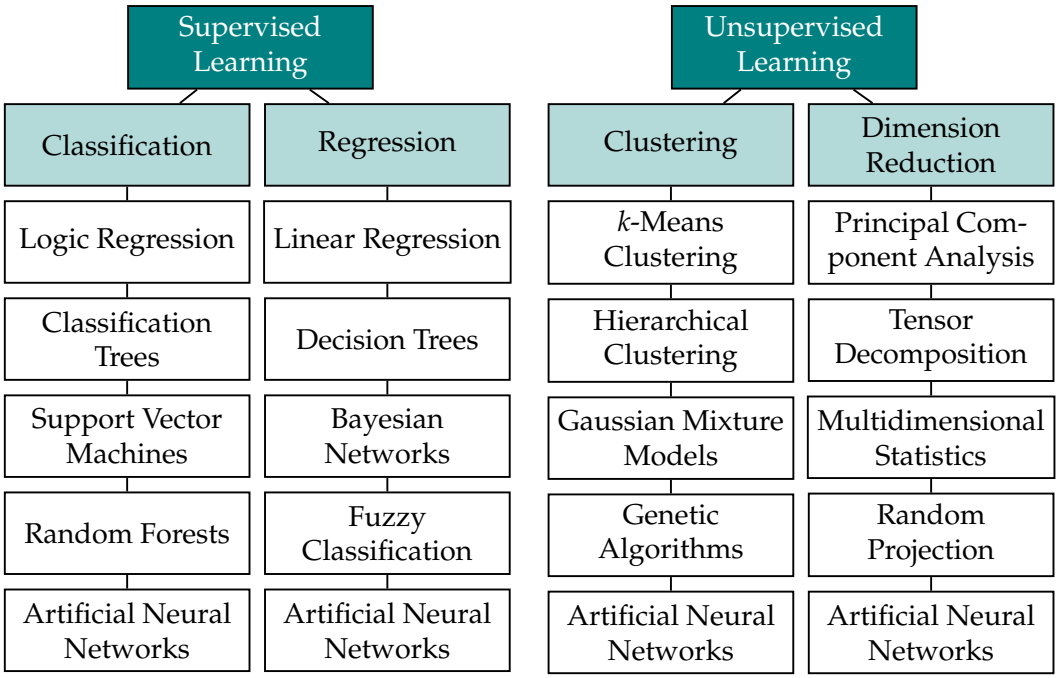


Figure 4.1: Overview of machine learning algorithms by categories. (Louridas and Ebert (2016))

As with all models and algorithms, each one has their own advantages and disadvantages with regards to solution quality, computation time (during training and prediction), complexity, robustness, transparency etc. Due to their "black

box character", it is not always known in advance which ML tool will perform best for a given task. Therefore, it might be useful to quick-train different models for comparison and then further optimize the most promising ones. Furthermore, fusion methods can be applied to yield better solutions. In the context of this work, such fusion methods combine the output of several ML tools to a unique prediction.The fusion techniques employed in the prediction model are the simple average, weighted average, majority voting, and the principal component analysis with multiple regression.

4.4 Machine Learning in Scheduling Literature

A wide variety of literature exists on ML in business applications. Often, these problems are related to forecasting or market predictions (e.g. demands, prices), eCommerce (e.g. customer segmentation, recommendations) or recruiting automation (e.g. candidate pre-selection). Usually, these problems are complex and include a lot of data, but differ largely from the combinatorial optimization structure of scheduling problems. Focussing on research related to combinatorial optimization and scheduling, around 100 papers are summarized in appendix A.1.

The ML research related to combinatorial optimization started with the work of Hopfield (1982) and Hopfield and Tank (1985), proposing a single layer recurrent neural network to solve a travelling salesman problem (TSP). The proposed recurrent NN consists of binary neurons (representing a permutation matrix) and an energy function that includes the objective function and penalty terms for constraint violations. This energy function is minimized by synchronous or asynchronous updates of neuron states until a equilibrium is reached and an optimized solution is found. This NN structure, known as "Hopfield Neural Network" (HNN), has been modified and adapted by many researchers afterwards. Akyol and Bayhan (2007) summarize the advantages and disadvantages of HNN as shown in table 4.2. All in all, they point out that the disadvantages of the approach led to the development of other methodologies to include ML in combinatorial optimization.

Table 4.2: Comparison of NN models for scheduling problems. (Akyol and Bayhan (2007))

Approach	Advantages	Disadvantages
HNN	• massive parallelism • convenient hardware implementation of the network architecture • applicable to different types of combinatorial optimization problems in various disciplines	• gets easily trapped in local minimum states • may not converge to good quality solutions • determining the appropriate values of the penalty and network parameters, and initial states is difficult and based on a trial-error process • trade-off occurs between the penalty terms to be minimized • way of incorporating constraints into the energy function affects the quality of the solution • termination criteria affect the quality of the results • translation of the problem into the energy function is difficult • network size grows with the problem size
BPN	• universal approximators • superior to competing ML tools in capturing the complex relationship between input and output variables of the considered scheduling problem • easy to be utilized in production scheduling, a trained network can produce its output very rapidly and thus justifies time required for training	• gradient based training techniques have the risk of local minima solutions • initial connection weights are a crucial issue to reduce the possibility of being trapped in local optimum • generating a training set is time consuming • generalization ability depends on the adequacy of training set • overfitting possibility • not really indicated for combinatorial optimization • backpropagation algorithm's robustness and speed are sensitive to its control parameters
Competitive NN	• best applicable to optimization and classification problems • penalty terms are handled explicitly by competitive learning rule, therefore the energy function is simplified and the time required in obtaining coefficients is reduced	• equations of motions need to be derived before solving the problem • not applicable for all scheduling problems
Hybrid Approaches	• the problems of convergence, stability, penalty parameter determination and sensitivity to the initial inputs may be overcome • solution quality may be increased and computational time decreased • integrating global search techniques can help to obtain global optimum solutions • advantages of each technique can be combined to overcome the shortcomings • compete effectively with other heuristics • adaptability to a dynamic environment is possible • evolutionary design of NN eliminates trial and error work of manual design of NN • by evolution of connection weights, the shortcomings of gradient descent based training algorithms may be overcome • NN complexity can be decreased and its generalization can be increased by evolutionary algorithms • evolutionary training can be faster and more reliable than back-propagation • evolutionary learning can be applied to problems where gradient information is unavailable or costly	• disadvantages of the individual methodologies may be encountered • evolutionary training may be slow for some problems • employing evolutionary algorithms at any level of evolution is time consuming • two major problems in evolving NN: noisy fitness evaluation problem and the complexity of permutation problem

Based on these disadvantages, competitive NN and multilayer perceptrons, mainly feedforward NN with backpropagation learning (BPN), gained popularity. Competitive NN are unsupervised NN (e.g. Kohonen's self-organizing map (SOM) (Kohonen (1982))), where the "winner-take-all" strategy forms the basis of the networks. A single layer of output neurons is fully connected to the input neurons of the network. Lateral inhibition occurs among the neurons in the output layer, and each neuron tries to inhibit the neuron to which it is laterally connected. For a given input pattern, the neuron with the (input) weight vector at the least distance from the input vector is called the winner and its output is set to one.
BPN consist of an input layer, one or more hidden layers and an output layer, and neurons are connected to (only) the next layer neurons by weighted links, updated by a kind of gradient descent technique with backward error propagation, evaluating the difference between actual and desired outputs (supervised learning). Advantages and disadvantages of these approaches can also be seen in table 4.2. (Akyol and Bayhan (2007))
Although pure ML approaches for solving special cases of combinatorial optimization problems exist, the focus of research shifted to hybrid approaches during the last decades because the combination of classical operations-research-algorithms with ML tools proved to be able to overcome the shortcomings of the independent approaches.
Min et al. (1998) describe some applications of NN in hybrid scheduling approaches as follows:

- NN can capture complex relationships between the input and output variables that are difficult to analytically relate such as the relationship between the performance measures and operational policy of a manufacturing system or between the job characteristics and the performance measure of a scheduling system. After learning the unknown correlation between the input and output data, they can generalize to predict or classify for the cases they were not exposed to.
- In some cases of production system design, NN can replace time consum-

ing simulation approaches.

- BPN can produce a schedule for a given set of input parameters (but they perform optimization only indirectly by learning from optimized training data).
- BPN and other ML tools are also used to select scheduling rules, optimize (meta-)heuristic parameters, calculate fitness values during metaheuristic iterations, or to estimate the system performance measures.
- Since real manufacturing environments are dynamic, flexible scheduling methods are needed to react to changes of the system state. Thus, in dynamic scheduling environments, NN are employed to reduce the need for rescheduling.
- While optimizing NN (e.g. HNN) are involved directly in the optimization by mapping the scheduling objective functions and constraints onto these networks, competitive NN can detect regularities and correlations in input vectors and adapt future responses accordingly.

The development of different research streams is partly mirrored by the number of publications found over time (see 4.2). Furthermore, the literature review makes apparent that the majority of authors apply NN in their scheduling approaches, whereas other ML tools, like SVM, are highly under-represented. Regarding to the number of publications, decision trees (DT) come next to NN, but they are predominantly applied in the field of dispatching rule (DR) detection or selection. With regards to real world problems, they are popular due to their solution transparency and their easy implementation. The selection of dispatching rules based on the current system state is one of the most widely used applications of ML tools in general. An overview of the distribution of applied ML algorithms and their respective output, or prediction function, is given in figures 4.3 and 4.4.

With regards to the production environment under consideration, the majority of papers considers flexible manufacturing systems (FMS) or job shops (see figure 4.5). An extensive overview on ML applications in scheduling is presented in chronological order in table A.1, where a brief summary of the investigated

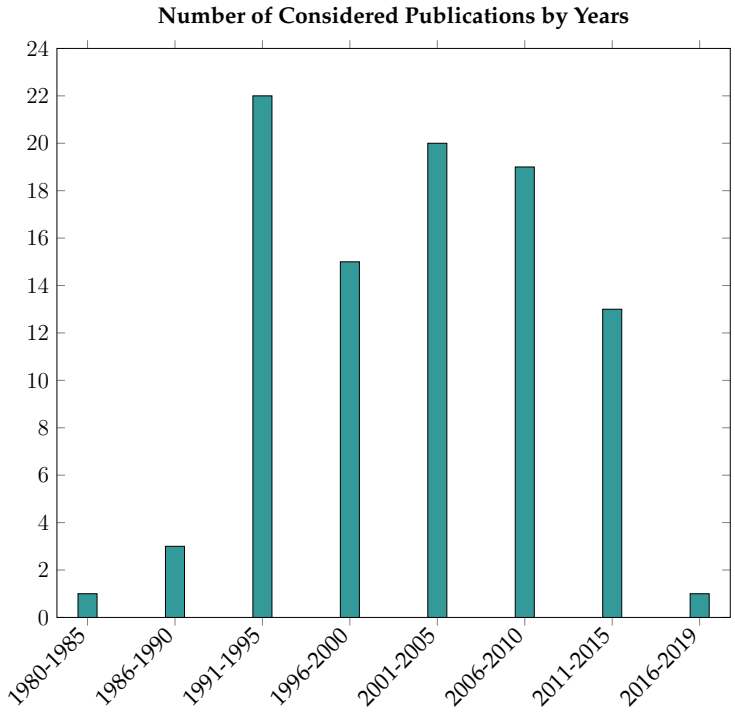


Figure 4.2: Chronological overview of ML applications in scheduling.

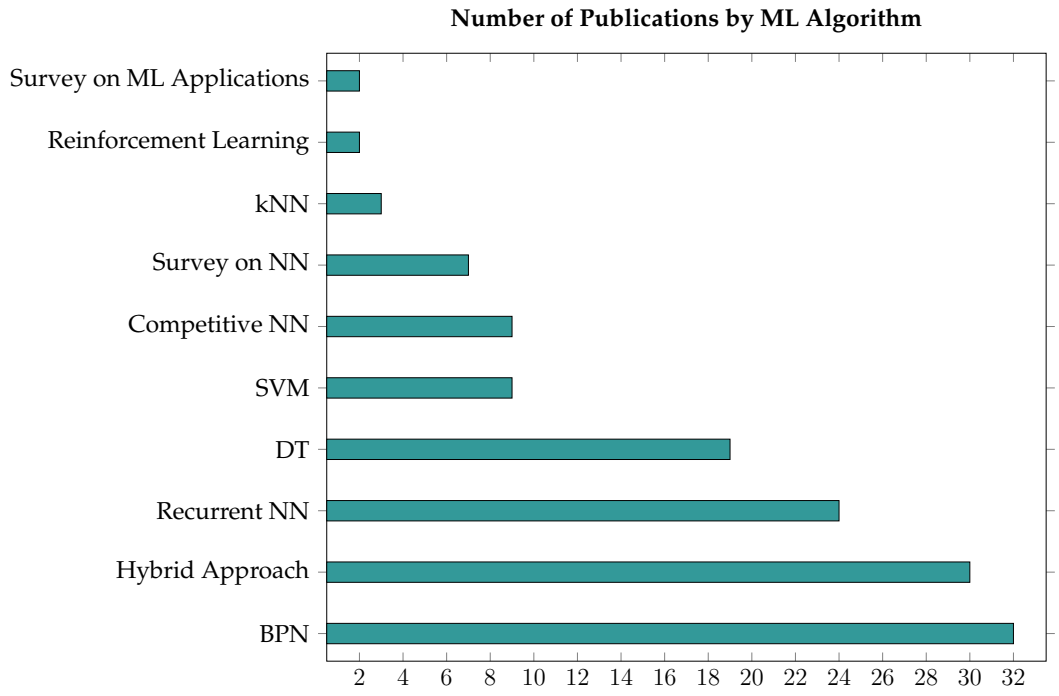


Figure 4.3: Overview of ML applications in scheduling by algorithms.

problems, further information on the type of ML algorithm applied, the optimization problem investigated, the objective and the actual specific output of the ML algorithm are provided.

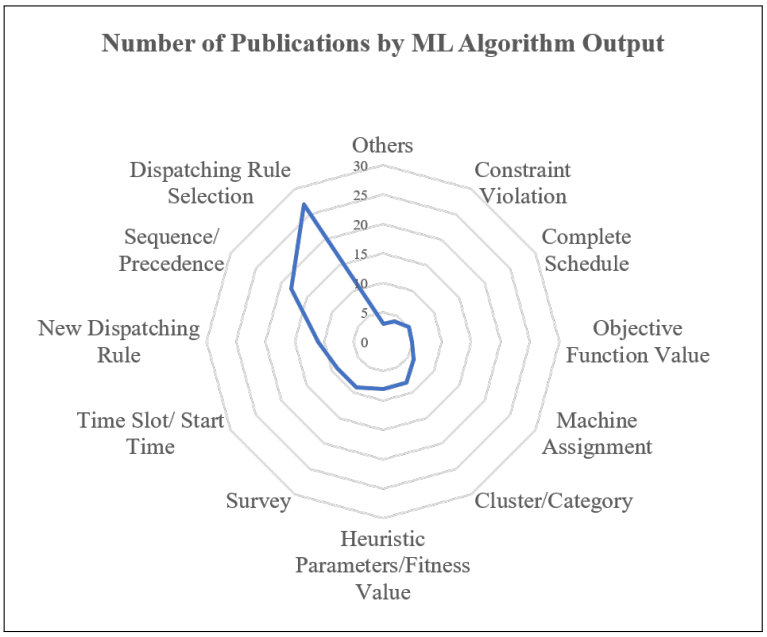


Figure 4.4: Overview of ML applications in scheduling by output of ML algorithms.

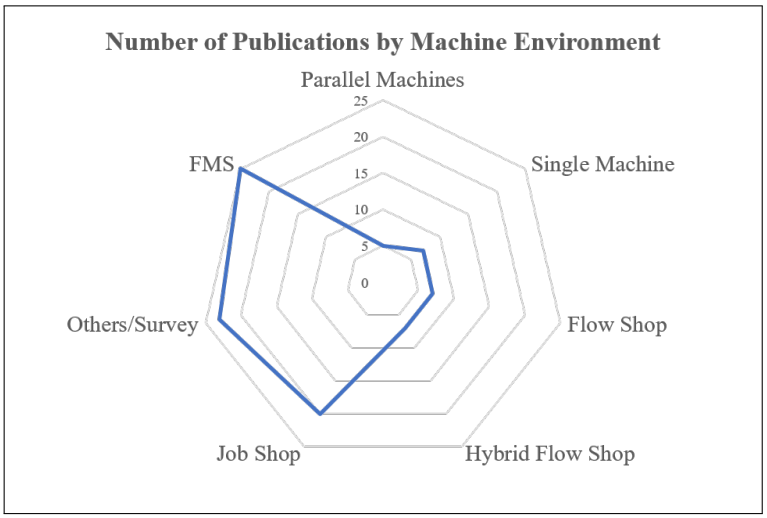


Figure 4.5: Overview of ML applications in scheduling by production system.

5 Hybrid Flow Shop Scheduling Approach

Based on the literature reviews presented in sections 4 and 5 it can be stated that a research gap exists with regards to flexible approaches for HFS with unrelated machines and flowtime minimization. Additionally, the application of ML techniques HFS scheduling problems has rarely been considered and the majority of work combining ML and scheduling deals with dispatching rules. With this work, we are aiming at partly covering both open research areas. Therefore, the formal problem formulation and solution approaches are presented in this section.

5.1 Problem Formulation

Using the three field notation by Vignier et al. (1999) the model under consideration can be classified as a $\mathrm{FH}_m((\mathrm{RM}^k)_{k=1}^m \mid prmu \mid C_{\max} \vee \overline{F})$ problem. The objective is either the minimization of makespan ($C_{\max}$) or flowtime ($\overline{F}$) while on each stage k a set RM^k of non-identical machines is available. As jobs are assumed to be available at time zero, total flowtime minimization equals minimizing the sum of job completion times.

The classical HFS scheduling problem assumes machines to be identical if they belong to the same stage. This more specific problem was proven to be NP-complete for $C_{\max}$-minimization by Lee and Vairaktarakis (1994) as it includes the special cases of the parallel machine problem (P_m-problem) if we only consider one stage as well as flow shop problems with at least three stages but only one machine per stage (F_3-problem). Both problems have been proven to be NP-complete by Graham et al. (1979).

From a computational complexity perspective, $\overline{F}$ is a more complex objective function than $C_{\max}$ and even the two-stage FS-problem is known to be NP-complete (Graham et al. (1979)). The $\mathrm{FH}_m((\mathrm{RM}^k)_{k=1}^m \mid prmu \mid C_{\max} \vee \overline{F})$ problem also includes the special case of identical machines so that NP-completeness can be inferred. This makes it necessary to develop approximation algorithms to solve medium and large instances, as only very small instances

can be solved to optimality within reasonable computation time. Nevertheless, relaxed solutions (e.g. limitation of computation time or relaxation of binary conditions) can be helpful to validate the performance of approximation algorithms.

We have modified the position-based mixed integer linear programming (MILP) model by Naderi et al. (2014) to solve the HFS problem with unrelated machines. For a given job list and processing times matrix, the optimal sequence of jobs, which remains the same on all m stages, and the optimal machine assignments for all jobs j at stages 1 to m are determined with respect to the chosen objective function.

5.1.1 Mathematical Model

The following notation is used to formulate the MILP model for the $\mathrm{FH}_m((\mathrm{RM}^k)_{k=1}^m \mid prmu \mid C_{\max} \vee \overline{F})$ problem, which will be used for comparison:

Parameters and Sets:

J	set of jobs
I	set of stages
L	set of machines
n_J	number of jobs in J
n_I	number of stages in I
n_L	total number of machines in L
M^i	subset of machines at stage i
$P_{j,l}$	processing time of job j on machine l
$\mathsf{big}M$	sufficiently large number

Indices:

j	$= 1, ..., n_J$	job index
k	$= 1, ..., n_J$	position in sequence
i	$= 1, ..., n_I$	stage index
l	$= 1, ..., n_L$	machine index

Decision Variables:

$T_{j,i}$	starting time of job j on stage i
$C_{j,i}$	completion time of job j on stage i
$S_{i,k}$	starting time of job in position k on stage i
$CS_{i,k}$	completion time of job in position k on stage i
$C_{\max}$	maximum completion time (Makespan)
$\overline{F}$	sum of completion times of all jobs on last stage (total Flowtime)

$$X_{j,k} = \begin{cases} 1 : \text{job } j \text{ is in position } k \\ 0 : \text{otherwise} \end{cases}$$

$$Y_{i,k,l} = \begin{cases} 1 : \text{job in position } k \text{ is assigned to machine } l \text{ at stage } i \\ 0 : \text{otherwise} \end{cases}$$

Objective functions to be chosen:

$$\min \quad C_{\max}, \tag{1}$$

$$\min \quad \overline{F}. \tag{2}$$

Subject to:

$$\sum_{k=1}^{n_J} X_{j,k} = 1 \qquad \forall j \in J, \tag{3}$$

$$\sum_{j=1}^{n_J} X_{j,k} = 1 \qquad \forall k \in J, \tag{4}$$

$$\sum_{l \in M^i} Y_{i,k,l} = 1 \qquad \forall i \in I, \forall k \in J, \tag{5}$$

$$T_{j,i+1} \geq T_{j,i} + P_{j,l} - \mathbf{big}M \cdot (2 - X_{j,k} - Y_{i,k,l}) \qquad \forall i = 1, ..., n_I - 1, j, k \in J, l \in L, \tag{6}$$

$$S_{i,k} \geq S_{i,t} + \sum_{j=1}^{n_J} X_{j,t} \cdot P_{j,l} - \mathbf{big}M \cdot (2 - Y_{i,k,l} - Y_{i,t,l}) \qquad \forall i \in I, l \in L, j, k, t \in J, k > 1, t < k, \tag{7}$$

$$S_{i,k} \geq T_{j,i} - \mathbf{big}M \cdot (1 - X_{j,k}) \qquad \forall i \in I, j, k \in J, \tag{8}$$

$$S_{i,k} \leq T_{j,i} + \mathbf{big}M \cdot (1 - X_{j,k}) \qquad \forall i \in I, j, k \in J, \tag{9}$$

$$C_{j,i} \geq T_{j,i} + P_{j,l} - \mathbf{big}M \cdot (2 - X_{j,k} - Y_{i,k,l}) \qquad \forall i \in I, l \in L, j, k \in J, \tag{10}$$

$$CS_{i,k} \geq S_{i,k} + \sum_{j=1}^{n_J} X_{j,k} \cdot P_{j,l} - \mathbf{big}M \cdot (1 - Y_{i,k,l}) \qquad \forall i \in I, l \in L, k \in J, \tag{11}$$

$$C_{\max} \geq max(C_{j,n_I}), \tag{12}$$

$$\overline{F} \geq \sum_{j=1}^{n_J} C_{j,n_I}, \tag{13}$$

$$X_{j,k} \in \{0, 1\} \qquad \forall j, k \in J, \tag{14}$$

$$Y_{i,k,l}, \in \{0, 1\} \qquad \forall i \in I, k \in J, l \in L, \tag{15}$$

$$T_{j,i} \geq 0 \qquad \forall j \in J, i \in I, \tag{16}$$

$$C_{j,i} \geq 0 \qquad \forall j \in J, i \in I, \tag{17}$$

$$S_{i,k} \geq 0 \qquad \forall i \in I, k \in J, \tag{18}$$

$$CS_{i,k} \geq 0 \qquad \forall i \in I, k \in J. \tag{19}$$

By including either (1) or (2), the objective function to be minimized can be selected. Constraint (3) ensures that every position is only assigned once and constraint (4) enforces that every job is only placed in one position. Constraint (5) selects exactly one machine out of the M^i machines available per stage i for every job. Constraints (6) and (7) ensure that no job can start being processed before it is completed at the previous stage and the machine has completed the previous job of the sequence. With constraints (8) and (9) the reference between job numbers and positions is implemented. Constraints (10) and (11) guarantee that the completion times of jobs equal at least the starting time plus respective processing time on all stages. Finally, constraints (14) to (19) define the decision variables.
Naderi et al. (2014) were able to solve some problems with up to 12 jobs and 4 stages with 1 to 3 machines per stage (for the identical machines case) while being faster than other comparable models for larger instances. As the unrelated machines case adds complexity to the model, we provide a relaxed solution by limiting the computation time to 3600 seconds. The modified version is able to solve 39 out of 1152 generated problems with up to 5 stages, 100 jobs and 7 machines per stage in the predefined computation time. The relaxed solutions are used as an upper bound for performance evaluation of our new heuristic and hybrid approach.

5.1.2 Testbed

Existing testbeds for (HFS) scheduling problems usually consider uniformly distributed processing times, thus not representing realistic problems. In real-world manufacturing, processing times can be assumed to be normally distributed in most cases. (Hopp and Spearman (2011)) Therefore, we take the approach of Watson et al. (2002) as a basis for developing a representative testbed for different kinds of practical manufacturing processes.

Instance Generation We generate combinations of the following features of the HFS system:

- 3 to 5 stages,
- 1 to 7 machines per stage,
- 25, 50, 75, 100 jobs,
- normally distributed processing times with different mean and standard deviation (8 settings, job or machine correlation).

By combining all options, we derive a testbed with 192 instances each, for the 3, 4, and 5 stages case respectively.

Machine Configurations The number of machines per stage is randomly generated with the following characteristics per category:

- constant (2 machines on all stages),
- equal (all from same interval; ranges are: {1,2,3},{4,5},{6,7}),
- ascending (upstream stages contain less machines than downstream stages),
- descending (upstream stages contain more machines than downstream stages),
- hill (midstream stages contain more machines than up-/downstream stages),
- valley (midstream stages contain less machines than up-/downstream stages).

Structured Processing Times To provide somewhat representative instances for real-world problems, normally distributed processing times are generated following an approach by Watson et al. (2002), assuming job or machine correlation (see figure 5.1). In job-correlated cases, processing times are largely independent of the selected machine, whereas in machine-correlated cases, job processing times are heavily depending on the machine they are assigned to. The actual values are generated randomly from normal distributions with different mean and standard deviations as shown in tables 5.1 and 5.2.

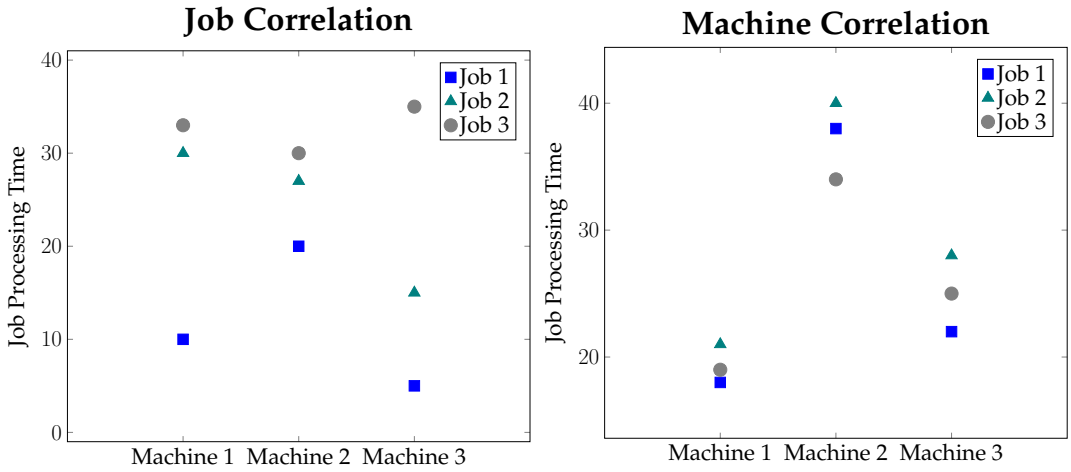


Figure 5.1: Examples of job and machine correlation for a three jobs and three machines setting. (Watson et al. (2002))

Table 5.1: Processing times (per job) of job correlation case.

Job Correlation Case	Mean of Processing Times	Standard Deviation of Processing Times
Equal	30	3
Diverse	30	25
Job Families: 1. Third	30	6
Job Families: 2. Third	50	10
Job Families: 3. Third	80	16

Table 5.2: Processing times (per stages) of machine correlation case.

Machine Correlation Case	Mean of Processing Times	Standard Deviation of Processing Times
Equal	50	10
Ascending	{30, 50, 80}	{6, 10 , 16}
Descending	{80, 50, 30}	{16, 10 6}
Hill	{30, 80, 30}	{6, 16, 6}
Valley	{80, 30, 80}	{16, 6,16}

5.1.3 Benchmarks

To evaluate the performance of the proposed method, we furthermore consider different lower and upper bounds. The well-known heuristic by Nawaz et al. (1983) is modified to provide the first upper bound for the HFS-problem as it was found to perform well for makespan as well as flowtime objective (see Brah and Loo (1999)).

As mentioned above, the results of the MILP-model described in section 5.1.1 are generated in PyCharm 2018.2 using the Gurobi 8.0.1 solver with a limited computation time of 3600 seconds. These results are taken as an additional upper bound for both objective functions.[10]

Lower bounds for makespan are developed based on Hidri and Haouari (2011). An overview of the number of publications mentioning lower bounds for HFS problems with makespan or flowtime objective is given in table 5.3. Whereas 88 % of those papers consider makespan minimization, only 12 % try to derive bounds for flowtime objectives, mainly in the context of stopping criteria in branch and bound algorithms.

Table 5.3: Number of reviewed papers published between 1987 and 2017 describing lower bounds for makespan or flowtime according to number of stages and type of parallel machines.

No. of Stages	**Machine Environment**			**Total**
	Identical	**Uniform**	**Unrelated**	
2	20	3	2	25
3	2	1	0	3
m	13	5	0	18
Total	**35**	**9**	**2**	**46**

Lower Bounds for Minimizing Makespan

Hidri and Haouari (2011) describe stage-based bounding strategies for the $\mathrm{FH}_m((\mathrm{PM}^k)_{k=1}^m \mid C_{\max})$ problem and propose a composite lower bound.[11] We

[10] Due to the time constraint, an upper bound is generated for both objective functions. In 39 out of 1152 instances, an optimal solution is found within the time limit.

[11] They also propose a destructive lower bound based on the concept of revised energetic reasoning.

adapt this composite bound (which simultaneously considers different stage-based bounds) for our $\mathrm{FH}_m((\mathrm{RM}^k)_{k=1}^m \mid C_{\max})$ problem. Therefore, we modify the unrelated machine environment to an identical machine problem. Furthermore, we propose a simple lower bound for flowtime minimization for the same problem.

In this section we will use the following notation:

j	$=$	$1, ..., nJ$	job index
h, k, l	$=$	$1, ..., nK$	stage indices
i	$=$	$1, ..., m_k$	machine index
m_k			no. of identical machines at stage k
p_{ji}			processing time of job j on machine i
M^k			set of identical machines at stage k

The general bounding scheme for stage-based lower bounds introduced by Hidri and Haouari (2011) estimates minimal front waiting times α_i of machines at stages before they can start processing the first job as well as minimal idle times β_i after finishing processing of the last job. The total workload of machine i is denoted as γ_i and following relation holds, with $C^*_{\max}$ being the optimal makespan value:

$$\frac{1}{m_k} \cdot (\sum_{i=1}^{m_k} \alpha_i + \sum_{i=1}^{m_k} \beta_i + \sum_{i=1}^{m_k} \gamma_i) \leq C^*_{\max}, \tag{20}$$

where the total workload for a given stage k equals

$$\sum_{i=1}^{m_k} \gamma_i = \sum_{j=1}^{nJ} p_{kj}. \tag{21}$$

A lower bound on $C^*_{\max}$ can be therefore expressed as

$$\frac{1}{m_k} \cdot (A_k + \sum_{j=1}^{nJ} p_{kj} + B_k) \leq C^*_{\max}, \tag{22}$$

with A_k and B_k being valid lower bounds on $\sum_{i=1}^{m_k} \alpha_i$ and $\sum_{i=1}^{m_k} \beta_i$ respectively. The authors test different approaches to define A_k and B_k and derive symme-

try properties. They conclude that, given an instance nI with nK stages and nJ jobs, the symmetric instance I^{-1} with reverse order of stages has the same optimal makespan value as the initial instance. Therefore, the corresponding lower bound value on the minimal front waiting times of the symmetric instance, i.e. A_k^{-1}, is a valid value for the lower bound B_k. Finally, they combine the best performing approaches in their composite bound.

Santos, Hunsucker and Deal's Lower Bounds

One of the first stage-based approaches by Santos et al. (1995) considers the average processing time for a given stage and estimates the minimal time jobs spend in previous stages ("heads" or "left-side total processing times") and following stages ("tails" or "right-side total processing times").
The authors also note that a trivial lower bound on makespan is the total processing time of the job with longest total duration:

$$\text{LB}\,0 = \max_j(\sum_{i=1}^{nK} p_{ji}). \tag{23}$$

Santos et al. (1995) introduce the concept of LS-values referring to the left-side total processing times before stage k per job and RS-values, i.e. the right-side total processing times after stage k. The LSA- and RSA-lists contain the sorted LS- and RS-values in ascending order, so that $\text{LSA}_i/\text{RSA}_i$ denote the i-th smallest values. In Hidri and Haouari (2011) these LSA_i- and RSA_i-values equal the "head"- and "tail"-values in lower bound LB 1 and are computed as

$$r_{kj} = \begin{cases} \sum_{i=1}^{k-1} p_{ji}, & \text{if } k > 1, \\ 0, & \text{if } k = 1, \end{cases}$$

$$q_{kj} = \begin{cases} \sum_{i=k+1}^{nK} p_{ji}, & \text{if } k < nK, \\ 0, & \text{if } k = nK. \end{cases}$$

The i-th smallest values of all r_{kj}- and q_{kj}-values are denoted by $r_{k[i]}$ and $q_{k[i]}$

respectively and LB 1 is defined as follows:

$$\text{LB 1} = \max_{1 \leq k \leq nK} \lceil \frac{1}{m_k} \cdot (\sum_{i=1}^{m_k} r_{k[i]} + \sum_{j=1}^{nJ} p_{jk} + \sum_{i=1}^{m_k} q_{k[i]}) \rceil. \tag{24}$$

Lower Bounds Based on Shortest Processing Time

For the parallel machine problem at a given single stage with flowtime objective ($\text{P}_m \mid \sum C_j$)), the optimal solution can be found by applying the Shortest Processing Time (SPT) rule (see Bruno et al. (1974)). Based on the symmetry property of A_k and B_k, Hidri and Haouari (2011) propose a lower bound with

$$A_k^3 = m_k \cdot r_{k-1,[1]} + \text{SPT}_{k-1}(m_k), \tag{25}$$

$$B_k^3 = \text{SPT}_{k+1}(m_k) + m_k \cdot q_{k+1,[1]}, \tag{26}$$

where, for a given stage $k \in \{1, ..., nK\}$, SPT_k is defined as the vector of jobs' completion times of the SPT-rule solution for the parallel machine problem at stage k. The jobs are scheduled in ascending order of processing times at stage k and assigned to the earliest available machine. The sum of completion times of the first m_k jobs, that is the m_k jobs with minimal completion times, is denoted as $\text{SPT}_k(m_k)$. (SPT_0, SPT_{nK+1}, r_0 and q_{nK+1} are set to zero.)
The proposed bound is[12]

$$\text{LB 3} = \max_{1 \leq k \leq nK} \lceil r_{k-1,[1]} + \frac{1}{m_k} \cdot (\text{SPT}_{k-1}(m_k) + \sum_{j=1}^{J} p_{jk} + \text{SPT}_{k+1}(m_k)) \rceil. \tag{27}$$

The authors further modify LB 3 to derive a stronger lower bound by defining A_k and B_k as follows:

$$A_{hk}^4 = m_k \cdot r_{h[1]} + \text{SPT}_h(m_k) + \sum_{i=1}^{m_k} \sum_{s=h+1}^{k-1} p_{[i]s}, \tag{28}$$

$$B_{kl}^4 = \sum_{i=1}^{m_k} \sum_{s=k+1}^{l-1} p_{[i]s} + \text{SPT}_l(m_k) + m_k \cdot q_{l[1]}. \tag{29}$$

[12] LB 2 omitted as it is a weaker version of LB 3.

For a given combination of stages $h, k, l \in \{0, ..., nK | h < k, k \leq l\}$, the term $\sum_{s=h+1}^{k-1} p_{[i]s}$ denotes the i-th smallest inter-stage transfer time of jobs between stages h and k. (SPT_0, SPT_{nK+1}, r_0 and q_{nK+1} are set to zero. If $h = 0$ or $h = k - 1$, then $\sum_{i=1}^{m_k} \sum_{s=h+1}^{k-1} p_{[j]s} = 0$.)
Thereby, an additional lower bound is given by

$$\mathrm{LB}\,4 = \max_{0 \leq h < k \leq l \leq nK} \lceil \frac{1}{m_k} \cdot (A_{hk}^4 + \sum_{j=1}^{J} p_{jk} + B_{kl}^4) \rceil. \tag{30}$$

Lower Bounds Based on Generalized Shortest Processing Time

To tighten A_{hk}^4 further, the authors define availability times for each machine m_i ($i = 1, ..., m_h$) as $a_i \equiv r_{h[i]}$. Then, the generalized Shortest Processing Time (gSPT) rule can derive an optimal solution for the $(\mathrm{P}, \mathrm{NC}_{inc} \mid \sum C_j)$[13] problem at stage h (see Yalaoui and Chu (2006)), where identical parallel machines have different availability times but all jobs have identical release dates. The gSPT rule schedules jobs in ascending order of processing times at stage h and assigns them to the machine available next, taking into account given availability times. New definitions for minimum waiting times and idle times can therefore be expressed as

$$A_{hk}^5 = \mathrm{gSPT}_h(m_k) + \sum_{i=1}^{m_k} \sum_{s=h+1}^{k-1} p_{[j]s}, \tag{31}$$

$$B_{kl}^5 = \sum_{i=1}^{m_k} \sum_{s=k+1}^{l-1} p_{[j]s} + \mathrm{SPT}_l(m_k), \tag{32}$$

so that a new lower bound for makespan is defined as

$$\mathrm{LB}\,5 = \max_{1 \leq h < k < l \leq nK} \lceil \frac{1}{m_k} \cdot (A_{hk}^5 + \sum_{j=1}^{J} p_{jk} + B_{kl}^5) \rceil. \tag{33}$$

Here, gSPT_h is defined as the vector of jobs' completion times of the gSPT-rule solution for the parallel machine problem at stage h and $\mathrm{gSPT}_h(m_k)$ equals the

[13] NC_{inc} indicates that number of machines is non-decreasing with time. This means that, at each decision point, the number of available machines for the next planning interval is more than or equal to the maximum number of available machines up to the current point in time. (Schmidt (2000))

sum of completion times of the first m_k jobs, that is the m_k jobs with minimal completion times.

Lower Bounds Based on extended Shortest Remaining Processing Time

In a next step, release dates for each job $j \in \{1, ..., J\}$ are considered and set to $r_j \equiv r_{hj}$. For the resulting $(\mathrm{P} \mid r_j \mid \sum C_j)$) a relaxation based on job splitting is solved, allowing jobs to be processed on more than one machine at a time. The extended Shortest Remaining Processing Time (eSRPT) rule described in Yalaoui and Chu (2006) is used to derive a solution for a given stage. At any time, a job with the shortest remaining processing time among available jobs is simultaneously processed on all available machines until another job with strictly shorter remaining processing time than that of the job in process becomes available.

An additional lower bound on makespan with

$$A^6_{hk} = \mathrm{eSRPT}_h(m_k) + \sum_{i=1}^{m_k} \sum_{s=h+1}^{k-1} p_{[j]s}, \tag{34}$$

$$B^6_{kl} = \sum_{i=1}^{m_k} \sum_{s=k+1}^{l-1} p_{[j]s} + \mathrm{eSRPT}_l(m_k), \tag{35}$$

can therefore be obtained by

$$\mathrm{LB}\,6 = \max_{1 \le h < k < l \le nK} \lceil \frac{1}{m_k} \cdot (A^6_{hk} + \sum_{j=1}^{J} p_{jk} + B^6_{kl}) \rceil. \tag{36}$$

Here, eSPT_h is defined as the vector of jobs' completion times of the eSPT-rule solution for the parallel machine problem at stage h and $\mathrm{eSPT}_h(m_k)$ equals the sum of completion times of the first m_k jobs, that is the m_k jobs with minimal completion times.

Finally, a composite lower bound on makespan is proposed considering the best stage-based lower bounds simultaneously:

$$\mathrm{LB}_{C_{\max}} = \max\{\mathrm{LB}\,1, \mathrm{LB}\,3, \mathrm{LB}\,4, \mathrm{LB}\,5, \mathrm{LB}\,6\}. \tag{37}$$

MILP

As only very small instances can be solved to optimality, the MILP-model can only provide an upper bound for larger problems by limiting the computation time and comparing the thereby determined relaxed solution to the results provided by the new approach. Hereby, makespan objective seems to be easier to solve than flowtime objective.

Nawaz-Enscore-Ham Heuristic

Nawaz et al. (1983) proposed a simple polynomial heuristic for makespan minimization in static-deterministic permutation flow shops. NEH is still one of the best and widely used heuristics for these problems (see for example Framinan et al. (2003)) and many modifications have been introduced. The original NEH heuristic first generates an initial order of jobs sorted by decreasing total processing times. Then, the first two jobs are scheduled based on the minimal (partial) makespan. An iterative insertion of jobs into a partial sequence according to the initial order of jobs is executed until all jobs are scheduled.
In HFS problems, the machine assignment of jobs on stages has to be considered as well. Therefore, we use a modified version, the NEH^{HFS} heuristic, with the following steps:

Step 1: For each job j select machine $l \in \text{RM}^i$ with the smallest processing time p_{jl} on stage i.

Step 2: For each job j calculate the total processing time, i.e. the sum of processing times on all stages i.

Step 3: Sort jobs in descending [ascending] order of total processing times.

Step 4: Pick jobs in the first and second position of the sorted list of step 3 and find best sequence for these two jobs by calculating makespan [flowtime] for the two possible sequences. The relative positions of these two jobs are fixed for the remaining steps of the algorithm. Set $i = 3$.

Step 5: Pick the job in the i-th position of the list generated in step 3 and find the sequence with minimal makespan [flowtime] by placing it at all possible i positions in the partial sequence found in the previous step. The relative positions of already assigned jobs to each other is fixed. The number of enumerations at this step equals i.

Step 6: Stop, if $i =$ no. of jobs, otherwise set $i = i + 1$ and go to step 5.

Modifications have been made regarding the selection of the fastest machine for each job at each stage as well as considering both ascending and descending order of total processing times.

5.2 Heuristic Based on Divide et Impera

The proposed Modular-Assignment-of-Jobs-heuristic (`MAJ/MAJ-LS` heuristic) is based on the well-known *divide et impera* strategy. The problem of sequencing and assigning jobs to machines is divided into smaller sub-problems (see figure 5.2). First, the sequencing problem is decomposed into smaller problems by considering job packages instead of single jobs. For a given list of jobs and a given batch size, all possible subsets of jobs of the respective batch size are evaluated.

In a next step, the problem of finding the best machine assignments for jobs in a job package is further decomposed into stage-wise solutions. Those individual solutions are then heuristically merged to an overall machine assignment for each job package.

In the next phase, different job package solutions are combined heuristically

to find the best disjoint combination of job packages and their permutation. Based on the selected job packages, the overall schedule can then be derived. Optionally, two local search routines (`HFS_LocalSearch` and `HFS_2opt`) can be performed. The first local search routine enhances the `NEH` insertion scheme with job-index-based insertion and swap routines and the second one is based on the `2-Opt` algorithm first proposed by Croes (1958) for the travelling salesman problem.

To give an example of the amount of complexity reduction by decomposition, a small HFS with 9 jobs and 3 stages containing 3 machines each is considered. If all jobs are sequenced individually, $2.77 \cdot 10^{18}$ possible solutions to the scheduling and machine assignment problem (even if assuming the same job sequence on all three stages is kept) exist. If only batches, or job packages, containing 3 jobs have to be scheduled, only 9,920,232 solutions have to be evaluated.

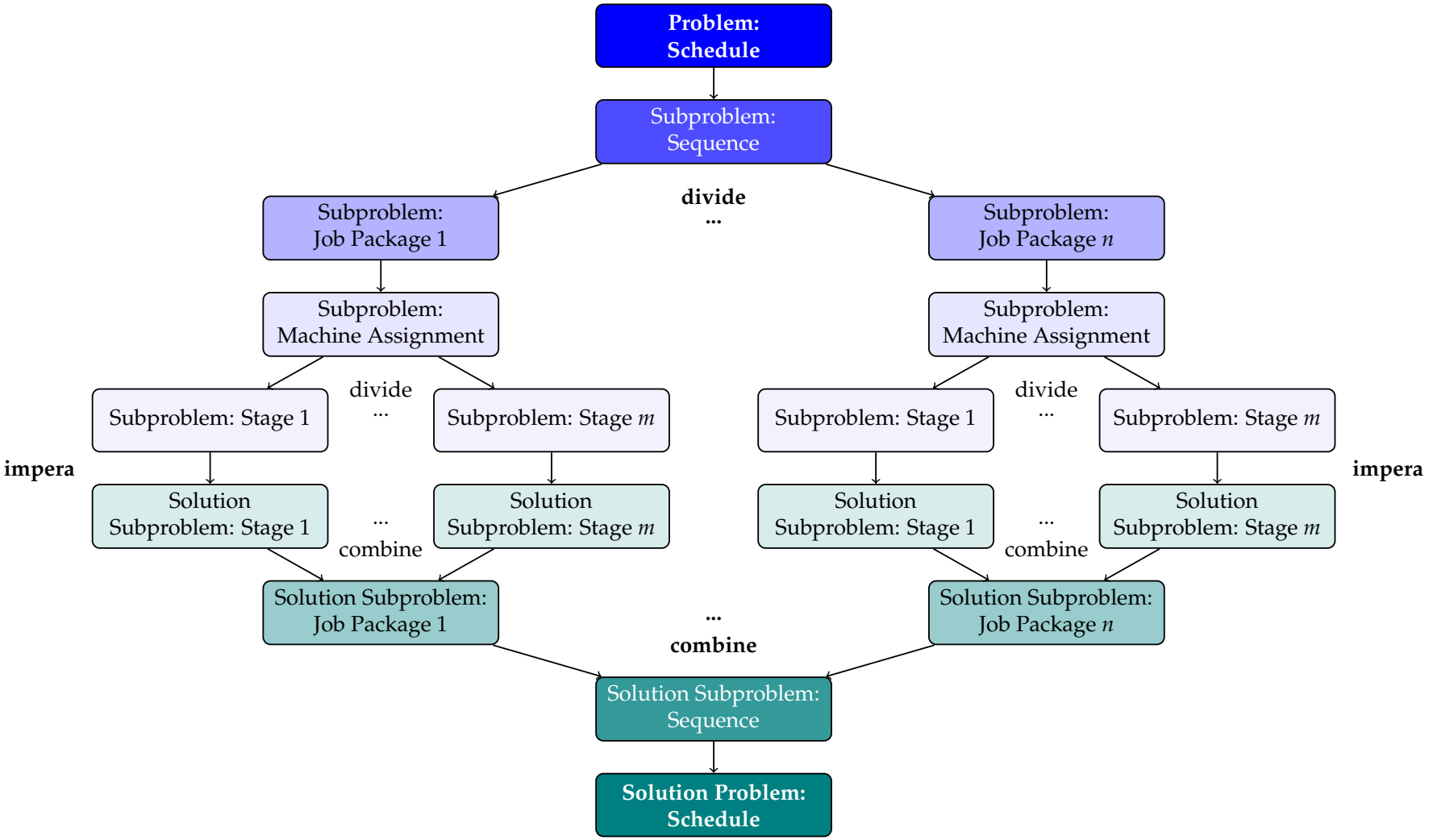


Figure 5.2: Divide et impera-based heuristic MAJ.

5.2.1 MAJ Heuristic Algorithm

The MAJ heuristic is modularly implemented in MATLAB. A flow diagram of function calls of MAJ is shown in appendix A.1. To start an optimization run, either a graphical user interface (GUI) can be used, as shown in appendix A.2, or the main function `HFS_Scheduler` can be called directly, which embeds all required sub-functions. The structure of `HFS_Scheduler` is presented in algorithm 1. For a given job processing time matrix (`machineTimes`) and stage configuration (`machineStages`), a close-to-optimal schedule is provided and a Gantt chart visualizes the schedule (see appendix A.3).
An optional preprocessor may reduce the number of possible solutions by eliminating unreasonable machine assignments beforehand. Depending on the structure of job processing times it might be useful, with regards to computation times, to exclude some solutions before starting the computation. This can be done by activating the preprocessor `HFS_ReduceMatrix` (setting `red` = 1), which heuristically excludes a certain amount of job/machine combinations based on structural indicators (see algorithm 2).
A major narrowing of the solution space is achieved by considering batching, so that not all jobs have to be sequenced one by one but only the optimal sequence of job packages has to be found. How many jobs those job packages contain (`nJobs`) may depend on the transport or production batch size of the company and has to be defined while calling the function `HFS_Scheduler`. The objective function to be minimized (`minimalValue`), is also selected in this step. Furthermore, it has to be decided whether or not major steps of the heuristic shall be parallelized (`par`).
Crucial functions are `HFS_JobCombinator` and `HFS_JobBundleCombinator`, as those are responsible for the "impera"-steps of MAJ following the "divide"-step (job package-wise and stage-wise decomposition). `HFS_JobCombinator` (see algorithm 3) calculates optimal machine assignments for each job package. In this most time-consuming and crucial step, only the best machine assignments per stage (plus a tolerance of `addStages`) are further combined to define the optimal overall machine assignments per job package.

Finally, a list of optimal machine assignments and corresponding objective function values for all possible job packages (based on the given job list) is stored in `packageMachineAssignments`. Notice that this list contains a maximum of $\binom{n}{nJobs}$ solutions with n being the total number of jobs. From this list, a disjoint combination of job packages has to be found so that all jobs are included and the fastest possible combination among these solutions is achieved. Therefore, the last sub-function `HFS_JobBundleCombinator` executes a heuristic routine of finding disjoint combinations and evaluating the corresponding objective values to determine the final schedule. The parameter `nJobPackages` hereby defines how many solutions are compared and highly influences the computation time as shown in algorithm 4.

Summarizing, if all parameters of `MAJ` would be set to their respective maximum possible value, a brute force calculation would be executed. By strategically limiting the solution space in every step of the heuristic, good solutions in reasonable time can be achieved. Executing `HFS_JobBundleCombinator` is the most time-consuming step so that accelerating it by applying machine learning algorithms is considered and will be presented in the next section.

In addition, two different improvement schemes can optionally be executed to further optimize the schedule found. Both schemes are based on local search routines and alter the sequence of jobs, keeping the machine assignment of the initial solution provided by `MAJ` unchanged.

Algorithm 1: MAJ (`HFS_Scheduler`)

Data: $machineTimes$, $machineStages$, $nJobs$, $minimalValue$, $addStages$, par, red, $nJobPackages$
Result: $makespan$, $flowtime$, $machineAssignment$
`/* Generates optimized machine assignment and sequence for a given processing time matrix to minimize makespan oder flowtime. */`
initialization;
if *total number of jobs is not a multiple of* $nJobs$ **then**
  add dummy jobs with zero processing times in $machineTimes$;
end
if *red = true* **then**
  reduce $machineTimes$ and determine $jobCombis$ with algorithm 2 (`HFS_ReduceMatrix`);
else
  $jobCombis \leftarrow \binom{n}{nJobs}$ = all possible subsets of total jobs with size $nJobs$ ($nJobs$ combination of n);
end
for $i \leftarrow 1$ **to** *no. of stages* **do**
  $machineAssignments(i) \leftarrow$ all possible machine assignments for $nJobs$ jobs at stage i (permutations with repetition);
  if *processing time structure* $\neq$ *case: job families, diverse* **then**
    reduce $machineAssignments$ by deleting combinations with more than $\lceil \frac{nJobs}{\text{machines at stage } i} \rceil$ jobs assigned to the same machine;
  end
end
`/* If par = true, the following loops are executed in parallel, else sequentially. */`
for $i \leftarrow 1$ **to** *no. of job packages* **do in parallel**
  $packageMachineAssignments(i) \leftarrow$ best machine assignment found by algorithm 3 (`HFS_JobCombinator`) for job package i;
  store job IDs, objective function value and machine selection;
end
Sort $packageMachineAssignments$ by ascending objective function value;
if $minimalValue$ = *makespan* **then**
  if $addJobPackages \leq \frac{\text{no. of job packages in } packageMachineAssignments}{2}$ **then**
    $selectedJobPackages \leftarrow$ first $nJobPackages$ and last $nJobPackages$ job packages in $packageMachineAssignments$;
  else
    $selectedJobPackages \leftarrow$ all job packages in $packageMachineAssignments$;
  end
else
  if $addJobPackages \leq \frac{\text{no. of job packages in } packageMachineAssignments}{2}$ **then**
    $selectedJobPackages \leftarrow$ first $2 \cdot nJobPackages$ job packages in $packageMachineAssignments$;
  else
    $selectedJobPackages \leftarrow$ all job packages in $packageMachineAssignments$;
  end
end
for $i \in selectedJobPackages$ **do in parallel**
  $minObjectives(i) \leftarrow$ best disjoint job packages combination for i-th selected job package, covering all jobs, found by algorithm 4 (`HFS_JobBundleCombinator`);
end
return $makespan$, $flowtime$, *and* $machineAssignment$ *for combination with minimal objective function value in* $minObjectives$

Algorithm 2: Reduce Time Matrix (`HFS_ReduceMatrix`)

Data: $machineTimes$, $machineStages$, $nJobs$
Result: $newMachineTimes$, $jobCombis$
`/* Eliminates specific job/machine combinations with high processing times if a sufficient number of alternatives exist. */`
initialization;
$maxDelete \leftarrow \lfloor \frac{\text{total no. of jobs}}{nJobs} \rfloor$;
for $i \leftarrow 1$ **to** *no. of stages* **do**
  if *no. of machines at stage* $i > 2$ **then**
    $Psort \leftarrow$ descendently sorted vector of processing times of jobs on machines at stage i;
    for $j \in Psort$ **do**
      if *total no. of deleted entries* $< maxDelete$ **and** *not more than* $\lfloor \frac{\text{available machines at stage } i}{2} \rfloor$ *entries deleted for an individual job* **then**
        $Psort(j) \leftarrow$ delete;
      end
    end
  end
end
$newMachineTimes \leftarrow machineTimes$, where deleted $Psort$ elements are replaced with -1, indicating blocked machines;
$jobCombis \leftarrow \binom{\text{total no. of jobs}}{nJobs}$;
return $newMachineTimes$, $jobCombis$

Algorithm 3: Machine Assignments (`HFS_JobCombinator`)

Data: $machineTimes$, $machineStages$, $nJobs$, $minimalValue$, $addStages$, $machineAssignments$
Result: $packageMachineAssignments$
/* Calculates the objective function value of different $machineAssignments$ for a given job package by first evaluating stage solutions and then combining them to an overall solution. Returns the best value found including the corresponding machine assignment matrix ($packageMachineAssignments$). */
initialization;
for $i \leftarrow 1$ **to** *no. of stages* **do**
  for $j \in machineAssignments(i)$ **do**
    if $minimalValue$ = *makespan* **then**
      $objValStage(i,j) \leftarrow$ array with corresponding machine selection and objective function value (makespan) for stage i;
    else
      $objValStage(i,j) \leftarrow$ array with corresponding machine selection and stage objective function value (flowtime) for stage i;
    end
  end
end
for $i \leftarrow 1$ **to** *no. of stages* **do**
  for $j \in machineAssignments(i)$ **do**
    delete $objValStage(i,j)$ if stage objective function value $>$ ($addStages$ + 1)-smallest stage objective function value found for stage i;
  end
end
$objValMin \leftarrow bigM$;
$possibleCombinations \leftarrow$ all possible combinations of remaining stage machine selections in $objValStage(i,j)$;
for $i \in possibleCombinations$ **do**
  if $minimalValue$ = *makespan* **then**
    $objValCombination \leftarrow$ makespan of i-th combination;
    if $objValCombination < objValMin$ **then**
      $objValMin \leftarrow objValCombination$;
    end
  else
    $objValCombination \leftarrow$ flowtime of i-th combination;
    if $objValCombination < objValMin$ **then**
      $objValMin \leftarrow objValCombination$;
    end
  end
end
return $packageMachineAssignments \leftarrow$ *array with* $objValMin$ *and corresponding machine assignment matrix*

Algorithm 4: Disjoint Job Package Combination (`HFS_JobBundleCombinator`)

Data: $packageMachineAssignments$, $selectedJobPackages$, $machineStages$, $nJobs$, $minimalValue$, $nJobPackages$
Result: $minObjectives$
`/* Combines different job packages to find an overall solution. A job package is taken from` *selectedJobPackages* `and disjoint job packages are added. */`
initialization;
for $i \in selectedJobPackages$ **do**
  $selectedSet \leftarrow selectedJobPackages(i)$;
  $disjointSet \leftarrow$ all job packages disjoint to job package i in $packageMachineAssignments$;
  $timeDiff \leftarrow$ vector of objective function difference between package i and packages in $disjointSet$;
  while *selectedSet does not include all jobs* **do**
    if $minimalValue$ = *makespan* **then**
      add job package with maximum $timeDiff$ to $selectedSet$ and remove from $disjointSet$;
    else
      add job package with minimum $timeDiff$ to $selectedSet$ and remove from $disjointSet$;
    end
    $timeDiff \leftarrow$ update;
  end
  if $minimalValue$ = *makespan* **then**
    $minObjectives(i) \leftarrow$ array including job packages information (job package makespans, machine assignments, and sequence) of $selectedSet$;
  else
    $minObjectives(i) \leftarrow$ array including job packages information (job package flowtimes, machine assignments, and sequence) of $selectedSet$;
  end
end
return $minObjectives$

5.2.2 Postprocessor Improvement Schemes

As indicated before, a given `MAJ` schedule may be further enhanced by two different improvement schemes. Algorithm 5 utilizes the `NEH` insertion scheme as well as job-index-based insertion and job-index-based swap scheme (Krishnaraj et al. (2012)) and algorithm 6 is based on the `2-Opt` method initially proposed by Croes (1958) for the travelling salesman problem.

First, the given job package sequence is taken as sorted list for $\texttt{NEH}^{\texttt{HFS}}$, which then first schedules the job packages in position one and two of this sorted list and then successively inserts the remaining job packages in all possible positions of partial sequences found in the previous iteration as described above, keeping the best found partial sequence in each iteration fixed for the next one.

The final sequence of the NEH step is taken as the initial solution for a local search using job-index-based insertion. Here, in each iteration (starting with the job package in first position up to the package in last position) a job package i is placed in all possible positions j of the initial sequence, retaining the order of all other packages. The best sequence found is stored and finally the best solution of all iterations is kept and taken as start solution for the job-index-based swap scheme. Here, instead of inserting a job package in position i at a position j, it is swapped with the job package currently placed at position j. Again, a best solution in each iteration is stored and the best out of these is kept as ultimate solution. Both insertion and swap scheme are repeated one more time.

The second improvement scheme employs the 2-Opt algorithm that was first developed for exchanging two edges in a tour of a travelling salesman solution and reassembling them in the other possible direction to compare the new total travel time or distance with the starting solution. (Croes (1958)) Here, job packages in a sequence are swapped and the new schedule and objective function value are determined in every iteration, starting with the job package in first position. The difference to the job-index-based swap method is that once an improved solution is found, the sequence is kept and the current iteration is restarted with the swapped job being considered for further steps. Therefore, the number of total iterations is not known in advance and might differ widely based on the quality of the initial solution.

Algorithm 5: Local Search Improvement Scheme (`HFS_LocalSearch`)

Data: $machineAssignment$, $nJobs$, $machineStages$, $minimalValue$
Result: $LSmakespan$, $LSflowtime$, $LSmachineAssignment$
/* Improves a given solution by altering the job package sequence through different local search schemes. */
initialization;
if $minimalValue$ = *makespan* **then**
  $ovValue \leftarrow$ makespan;
else
  $ovValue \leftarrow$ flowtime;
end
$PackageSequence \leftarrow$ Permutation of job packages in $machineAssignment$;
/* Step 1: NEH improvement scheme: */
$NEHstart1 \leftarrow$ Permutation of job packages in positions 1 and 2 in $PackageSequence$ with minimal $ovValue$;
$NEHseq \leftarrow NEHstart1$;
for $i \leftarrow 3$ **to** *no. of job packages* **do**
  for $j \leftarrow 1$ **to** i **do**
    insert job package in position i in $PackageSequence$ into positions j of $NEHseq$, keeping the partial sequence of remaining job packages unchanged;
    $NEHseq \leftarrow$ keep new permutation with minimal $ovValue$;
  end
end
$NEHov \leftarrow$ corresponding minimal $ovValue$;
/* Step 2: Job index-based insertion scheme: */
$JISseq \leftarrow NEHseq$;
$JISov \leftarrow NEHov$;
for $i \leftarrow 1$ **to** *no. of job packages* **do**
  for $j \leftarrow 1$ **to** *no. of possible positions in sequence* **do**
    if $i \neq j$ **then**
      insert job package in position i of $JISseq$ into position j, keeping the partial sequence of remaining job packages unchanged;
      $JISovIt(i,j) \leftarrow ovValue$ corresponding to current sequence;
    end
  end
  $JISovIt(i) \leftarrow$ permutation of job packages with minimal $ovValue$ in current iteration;
end
if *minimal* $ovValue$ *in* $JISovIt < NEHov$ **then**
  $JISseq \leftarrow$ sequence of job packages with minimal $ovValue$ in $JISovIt$;
  $JISov \leftarrow$ corresponding to minimal $ovValue$;
else
  $JISseq \leftarrow NEHseq$;
  $JISov \leftarrow NEHov$;
end
/* Step 3: Job index-based swap scheme: */
$JSSseq \leftarrow JISseq$;
$JSSov \leftarrow JISov$;
for $i \leftarrow 1$ **to** *no. of job packages* **do**
  for $j \leftarrow 1$ **to** *no. of possible positions in sequence* **do**
    if $i \neq j$ **then**
      swap job package in position i of $JSSseq$ with job package in position j, keeping the partial sequence of remaining job packages unchanged;
      $JSSovIt(i,j) \leftarrow ovValue$ corresponding to current sequence;
    end
  end
  $JSSovIt(i) \leftarrow$ permutation of job packages with minimal $ovValue$ in current iteration;
end
if *minimal* $ovValue$ *in* $JSSovIt < JISov$ **then**
  $JSSseq \leftarrow$ sequence of job packages with minimal $ovValue$ in $JSSovIt$;
  $JSSov \leftarrow$ corresponding to minimal $ovValue$;
else
  $JSSseq \leftarrow JISseq$;
  $JSSov \leftarrow JISov$;
end
/* Step 4: Repeat job index-based insertion scheme for currently best solution: */
$JIS2seq \leftarrow JSSseq$;
$JIS2ov \leftarrow JSSov$;
repeat job index-based insertion improvement and update $JIS2seq$ and $JIS2ov$;
/* Step 5: Repeat job index-based swap scheme for currently best solution: */
$JSS2seq \leftarrow JIS2seq$;
$JSS2ov \leftarrow JIS2ov$;
repeat job index-based swap improvement and update $JSS2seq$ and $JSS2ov$;
$LSmakespan \leftarrow$ makespan corresponding to $JSS2seq$;
$LSflowtime \leftarrow$ flowtime corresponding to $JSS2seq$;
$LSmachineAssignment \leftarrow$ machine assignment corresponding to $JSS2seq$;
return $LSmakespan, LSflowtime, LSmachineAssignment$

Algorithm 6: 2-Opt Improvement Scheme (`HFS_2opt`)

Data: $machineAssignment$, $machineStages$, $minimalValue$, $factorTolerance$
Result: $2optMakespan$, $2optFlowtime$, $2optMachineAssignment$
`/* Improves a given solution by altering the job package sequence through 2-Opt algorithm based local search scheme. */`
initialization;
$2optMachineAssignment \leftarrow machineAssignment$;
if *minimalValue = makespan* **then**
  $ovValue \leftarrow$ makespan;
else
  $ovValue \leftarrow$ flowtime;
end
$ovValue2opt \leftarrow ovValue$ corresponding to $machineAssignment$;
$count1 \leftarrow 1$;
$endsignal1 \leftarrow 0$;
while $endsignal1 = 0$ **do**
  $count2 \leftarrow count1 + 1$;
  $endsignal2 \leftarrow 0$;
  while $endsignal2 = 0$ **do**
    $machineAssignmentIt \leftarrow 2optMachineAssignment$;
    $machineAssignmentIt \leftarrow$ swap jobs in positions $count1$ and $count2$ in $machineAssignmentIt$;
    $ovValue2optIt \leftarrow ovValue$ corresponding to $machineAssignmentIt$;
    if $minimalValue =$ *makespan* **and** $ovValue2optIt < ovValue2opt$ **or** $ovValue2optIt < factorTolerance \cdot ovValue2opt$ **then**
      $2optMachineAssignment \leftarrow machineAssignmentIt$;
      $ovValue2opt \leftarrow ovValue2optIt$;
      $count1 \leftarrow 1$;
      $count2 \leftarrow count1 + 1$;
      $endsignal2 \leftarrow 1$;
    else
      $count2 \leftarrow count2 + 1$;
      if $count2 >$ *total no. of jobs* **then**
        $count1 \leftarrow count1 + 1$;
        $endsignal2 \leftarrow 1$;
      end
    end
  end
  if $count1 >$ *total no. of jobs* **then**
    $endsignal1 \leftarrow 1$;
  end
end
$2optMakespan \leftarrow$ makespan corresponding to $2optMachineAssignment$;
$2optFlowtime \leftarrow$ flowtime corresponding to $2optMachineAssignment$;
return $2optMakespan$, $2optFlowtime$, $2optMachineAssignment$

5.3 Hybrid Scheduling Approach

In the following sections, the general structure of the hybrid approach is outlined first, before the integrated ML model is explained in detail.

5.3.1 Heuristic Combining MAJ and Machine Learning

In practical applications, job processing times usually follow a certain distribution and do not change rapidly if there is no major change in the production process, machine layout or product portfolio. This led to the idea that the underlying pattern of optimal schedules can be learned and predicted by machine learning algorithms.

As mentioned before, a time-consuming step of `MAJ-LS` is the module `HFS_JobCombinator`. This step can be replaced by SVM and NN predictions to accelerate the heuristic. Initial computational tests show that the accuracy of those predictions is very high if sufficient training data is available. Furthermore, predictions can be made in negligible time, even for large instances, and the ML tools are able to extrapolate, meaning that training data for e.g. 20 jobs can be used to predict results for e.g. 80 jobs, if the underlying distribution of job processing times is not changed.

Algorithm 7 (`HFS_LearnDataGen`) executes the main steps of algorithm 1 (`HFS_Scheduler`) until for every possible job package of a given instance a best machine assignment is found. The corresponding results are then saved in vectorized form to provide a two-dimensional matrix for training NNs and SVMs, e.g. in `EIDOminer`. An example of such a training data set for n jobs and a job package size x is shown in table 5.4, where the objective function value for a given job package is to be learned, based on respective job processing times, and will afterwards be predicted by the trained ML regression model.

Algorithm 7: MAJ Training Data (`HFS_LearnDataGen`)

Data: $machineTimes$, $machineStages$, $nJobs$, $minimalValue$, par
Result: $learnData$
`/* Generates training data set for a given input instance and job package size. */`
initialization;
$jobCombis \leftarrow \binom{n}{nJobs}$ = all possible subsets of total jobs with size $nJobs$ ($nJobs$ combination of n);
for $i \leftarrow 1$ **to** *no. of stages* **do**
  $machineAssignments(i) \leftarrow$ all possible machine assignments for $nJobs$ jobs at stage i (permutations with repetition);
  if *processing time structure* $\neq$ *case: job families, diverse* **then**
    reduce $machineAssignments$ by deleting combinations with more than $\lceil \frac{nJobs}{\text{machines at stage } i} \rceil$ jobs assigned to the same machine;
  end
end
if *par* = *true* **then**
  for $i \leftarrow 1$ **to** *no. of job packages* **do in parallel**
    $packageMachineAssignments(i) \leftarrow$ best machine assignment found by algorithm 3 (`HFS_JobCombinator`) for job package i;
    store job IDs, objective function value and machine selection;
  end
  $learnData \leftarrow$ vectorized $packageMachineAssignments$ list;
else
  for $i \leftarrow 1$ **to** *no. of job packages* **do**
    $packageMachineAssignments(i) \leftarrow$ best machine assignment found by algorithm 3 (`HFS_JobCombinator`) for job package i;
    store job ID's, objective function value and machine selection;
  end
  $learnData \leftarrow$ vectorized $packageMachineAssignments$ list;
end
return $learnData$

Table 5.4: Example of training data set provided by algorithm 7.

Job Package	Processing Times Vector of Job at Position 1 in Package	...	Processing Times Vector of Job at Position x in Package	Objective Function Value
1	57.3 61.8 ... 20.4	...	49.9 62.2 ... 19.9	114.1
...	...	...	...	...
$\binom{n}{x}$	59.8 64.7 ... 18.3	...	56.2 68.9 ... 23.5	119.8

Notice that the objective function values alone do not provide a complete solution, as the corresponding machine assignment is not an output of the ML model. However, an important outcome of the decomposition-based approach is, that the ultimate solution does not depend on the exact stage-wise and job-package-wise solutions, but on their relation (i.e. the rank of sub-solutions). Therefore, only job packages with the best objective function values have to be considered further and for some best combinations of disjoint job packages an optimal machine assignment has to be found. This reduces the computation time by a large extent and is done by algorithm 8 (`HFS_Prediction`), which then outputs the overall solution for a given new (and previously unknown) problem instance.[14] For practical applications this means that once a job list of a given planning period has been considered as training data, the trained model can be directly applied during following planning periods. New training is only required e.g. if the production layout changes or if a major change in product portfolio or specifications occurs.

[14]Training data and the problem instance to be solved thereby have to belong to the same problem category, representing different production layouts and processes, as described in section 5.1.2.

Algorithm 8: MAJ-ML (`HFS_Prediction`)

Data: $machineTimes$, $machineStages$, $nJobs$, $minimalValue$, $addStages$, $project$, $analysis$, $nJobPackages$
Result: $makespan$, $flowtime$, $machineAssignment$
`/* Generates optimized machine assignment and sequence for a given processing time matrix to minimize makespan oder flowtime based on predicted objective function values for job packages. */`
initialization;
if *total number of jobs is not a multiple of* $nJobs$ **then**
  add dummy jobs with zero processing times in $machineTimes$;
end
$jobCombis \leftarrow \binom{n}{nJobs}$ = all possible subsets of total jobs with size $nJobs$ ($nJobs$ combination of n);
for $i \leftarrow 1$ **to** *no. of job packages* **do in parallel**
  $packageObjectiveValues(i) \leftarrow$ best objective function value predicted by `EIDOminer` prediction function defined by $project$ and $analysis$ for job package i;
end
sort $packageObjectiveValues$ by ascending objective function value;
for $i \leftarrow 1$ **to** *min(no. of job packages,* $nJobPackages$*)* **do**
  $selectedSet(i) \leftarrow packageObjectiveValues(i)$;
  $disjointSet(i) \leftarrow$ all job packages disjoint to job package i in $packageObjectiveValues$;
  $timeDiff \leftarrow$ vector of objective function difference between package i and packages in $disjointSet(i)$;
  while $selectedSet(i)$ *does not include all jobs* **do**
    if $minimalValue$ = *makespan* **then**
      add job package with maximum $timeDiff$ to $selectedSet(i)$ and remove from $disjointSet(i)$;
    else
      add job package with minimum $timeDiff$ to $selectedSet(i)$ and remove from $disjointSet(i)$;
    end
    $timeDiff \leftarrow$ update;
  end
end
for $i \leftarrow 1$ **to** *no. of stages* **do**
  $machineAssignments(i) \leftarrow$ all possible machine assignments for $nJobs$ jobs at stage i (permutations with repetition);
  if *processing time structure* $\neq$ *case: job families, diverse* **then**
    reduce $machineAssignments(i)$ by deleting combinations with more than $\lceil \frac{nJobs}{\text{machines at stage } i} \rceil$ jobs assigned to the same machine;
  end
end
`/* Machine assignments only have to be found for selected job packages. */`
for $i \leftarrow 1$ **to** *min(no. of job packages,* $nJobPackages$*)* **do**
  initialize processing time information for job packages in $selectedSet(i)$;
  for j *in packages in disjoint set* **do**
    $packageMachineAssignments(i, j) \leftarrow$ best machine assignment found by algorithm 3 (`HFS_JobCombinator`);
  end
  if $minimalValue$ = *makespan* **then**
    $minObjectives(i) \leftarrow$ array including job packages information, overall makespan, and best machine assignment for $selectedSet(i)$;
  else
    $minObjectives(i) \leftarrow$ array including job packages information, overall flowtime, and best machine assignment for $selectedSet(i)$;
  end
end
return $makespan$, $flowtime$, *and* $machineAssignment$ *for combination with minimal objective function value in* $minObjectives$

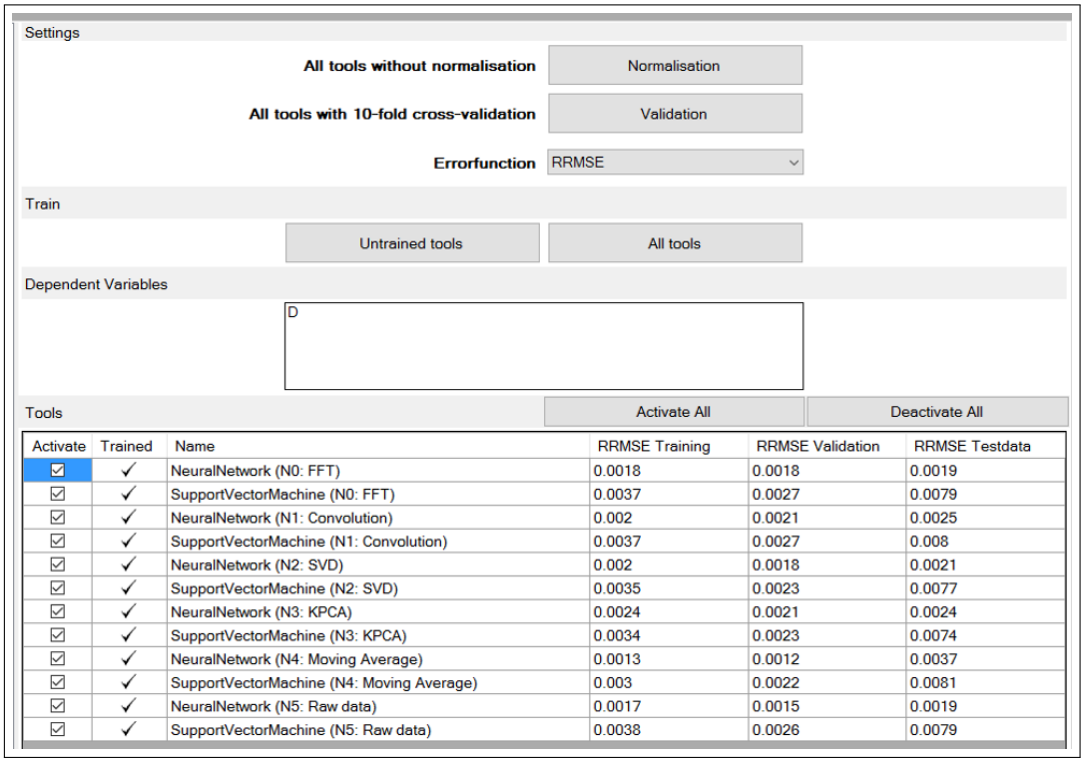
5.3.2 Machine Learning Predictions for Scheduling Approach

Generally, any ML environment can be used in combination with algorithm 7 (`HFS_LernDataGen`) and algorithm 8 (`HFS_Prediction`) to built a regression model whose prediction function can then be called by MATLAB. Here, all problems related to choosing the right ML algorithm and setting the respective hyperparameters accordingly appear as has been shown in chapter 4. We use the EIDOMINER software due to user-friendliness, availability of automated data preprocessing, parameter optimization, and fusion methods.
EIDOMINER features some data preprocessing steps that can be applied beforehand (also in combination) to improve the quality of training data, like convolution, fast fourier transformation, moving average, kernel principal component analysis, and singular value decomposition. Next, ML algorithms can be selected from following list:

- Bayesian networks,
- decision tree C4.5,
- classification and regression tree,
- kNN,
- multiple regression,
- BPN,
- SVM.

For the selected tools, z-score normalization can be activated, and the validation method (hold-out vs. k-fold cross validation) and error function (symmetric mean absolute percentage error (SMAPE) or relative root mean square error (RRMSE)) have to be chosen. Then, an automated "quick training" of all selected tools is performed. Based on those first results, most suitable ML algorithms are proposed (see figure 5.3) which can then be further optimized by either manual or automatic parameter optimization and/or additional training cycles.
Finally, the best performing individual algorithms are combined (either by majority voting, average, weighted average or by principal component analysis with multiple regression) to derive a final prediction function for unknown

data (see figure 5.4).[15] For the specific problem considered here, the prediction quality is usually very high, as is exemplary shown in figure 5.5. Here, more than 500 makespan values for the same number of job packages have been predicted with a coefficient of determination (R^2) of 0.9989.



Activate	Trained	Name	RRMSE Training	RRMSE Validation	RRMSE Testdata
☑	✓	NeuralNetwork (N0: FFT)	0.0018	0.0018	0.0019
☑	✓	SupportVectorMachine (N0: FFT)	0.0037	0.0027	0.0079
☑	✓	NeuralNetwork (N1: Convolution)	0.002	0.0021	0.0025
☑	✓	SupportVectorMachine (N1: Convolution)	0.0037	0.0027	0.008
☑	✓	NeuralNetwork (N2: SVD)	0.002	0.0018	0.0021
☑	✓	SupportVectorMachine (N2: SVD)	0.0035	0.0023	0.0077
☑	✓	NeuralNetwork (N3: KPCA)	0.0024	0.0021	0.0024
☑	✓	SupportVectorMachine (N3: KPCA)	0.0034	0.0023	0.0074
☑	✓	NeuralNetwork (N4: Moving Average)	0.0013	0.0012	0.0037
☑	✓	SupportVectorMachine (N4: Moving Average)	0.003	0.0022	0.0081
☑	✓	NeuralNetwork (N5: Raw data)	0.0017	0.0015	0.0019
☑	✓	SupportVectorMachine (N5: Raw data)	0.0038	0.0026	0.0079

Figure 5.3: Screenshot of EIDOMINER function box.

[15] Such a prediction function can be exported to other software applications, e.g. to commonly used spreadsheets.

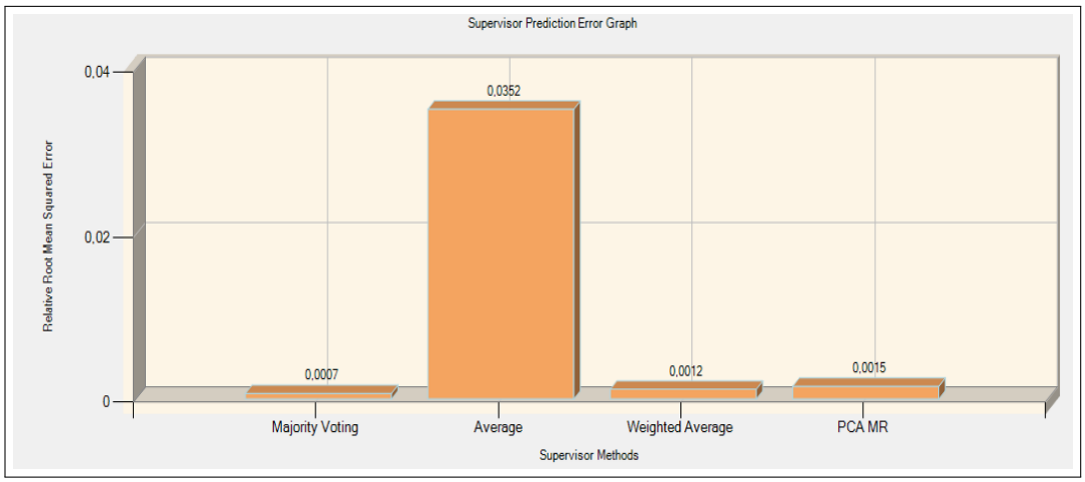


Figure 5.4: Screenshot of EIDOMINER fusion results.

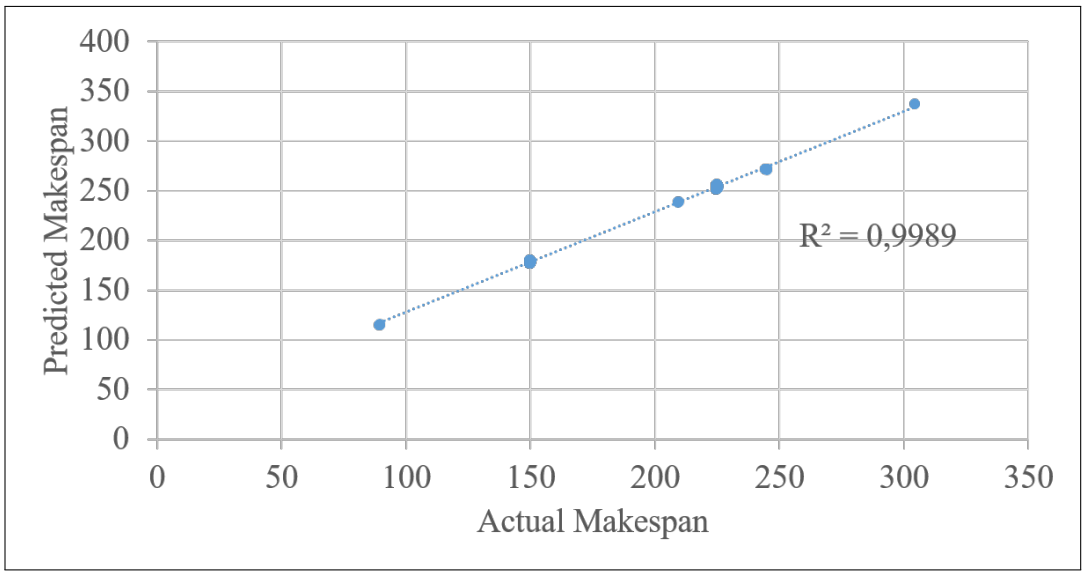


Figure 5.5: Example of EIDOMINER prediction function quality.

6 Results

In this section, the results of the numerical experiments are systematically presented, starting with the MAJ-approach, followed by MAJ−ML, and concluded by a stability analysis of both in comparison to NEH.

6.1 MAJ Heuristic

With a set of standard parameter configurations, an optimization routine is executed to calculate first results of algorithm 1 (HFS_Scheduler) and algorithm 5 (HFS_LocalSearch) for makespan and flowtime minimization on a representative testbed of $192 \cdot 3 = 576$ instances each. This combination will be further called the MAJ−LS-approach. A comparison with NEH results as well as lower and upper bounds is presented in table 6.1.

Table 6.1: Optimization routine results MAJ−LS.

MAJ−LS Approach	3 Stages		4 Stages		5 Stages	
Comparison	**Makespan**	**Flowtime**	**Makespan**	**Flowtime**	**Makespan**	**Flowtime**
MAJ−LS better than NEH	124	106	121	93	108	94
MAJ−LS same as NEH	53	0	0	0	50	0
MAJ−LS worse than NEH	15	86	71	99	34	98
Total no. of instances	192	192	192	192	192	192
Avg. deviation MAJ−LS to NEH (in %)	5.49	1.30	4.34	0.87	4.02	1.28
Std. dev. of deviation	7.48	3.08	7.04	3.45	6.52	3.92
Maximum improvement by MAJ−LS to NEH (in %)	37.90	14.85	38.47	13.91	38.97	23.35
Maximum deterioration by MAJ−LS to NEH (in %)	-1.46	-4.71	-12.82	-9.43	-10.46	-7.49
Avg. deviation MAJ−LS to MILP upper bound (in %)	3.45	12.63	4.11	12.9	9.1	19.18
Avg. deviation MAJ−LS to lower bounds (in %)	-3.25	NA	-4.34	NA	-5.26	NA
Total computation time (in h)	65.6	115.0	48.9	40.1	167.7	63.2

Keeping in mind that MAJ−LS is supposed to be configured to suit a given test instance by varying the heuristic parameters, the results are very promising. During the brute force routine, parameters were mostly kept constant and only the job package size (nJobs) was varied between 3, 4 and 5. Therefore, the results presented in table 6.1 do not represent optimal results of the proposed approach. Also, the results vary strongly amongst instances (observing the standard deviation of the average deviation between MAJ−LS and NEH results). This should not be the case if the same instance configuration (e.g. same number of jobs, stages, machines and same processing time structure) is solved multiple times with randomly generated processing times by MAJ−LS,

because, as it will be shown in chapter 6.3, the MAJ approach is robust with regards to solution quality.

Generally, makespan optimization leads to better results than flowtime minimization. The cases where MAJ−LS led to higher makespan values than NEH can be resolved with different parameter settings to improve the makespan further. Some random examples of how makespan can be decreased by varying the MAJ−LS parameters are presented in table 6.2. Eventually, MAJ−LS theoretically provides the option of a complete enumeration by setting all parameters accordingly. Naturally, this is only reasonable for small instances, but the algorithm itself is able to determine the optimal solution if computation time is neglected. This is not the case with NEH where only one solution is found and no option exists to influence the search procedure through parameters.

Table 6.2: Influence of parameter adjustment.

Instance	No. of Jobs	Machines per Stages	Makespan NEH	nJobs	addStages	addJobPackages	Makespan MAJ−LS	Difference MAJ−LS to NEH in %
I3,22	50	[5 2 4]	771.4	3	4	20000	775.4	0.5
I3,22	50	[5 2 4]	771.4	3	10	1000000	769.6	-0.2
I4,81	25	[1 4 5 3]	819.8	2	4	50000	825.9	0.7
I4,81	25	[1 4 5 3]	819.8	5	100	50000	801.5	-2.2
I5,141	75	[1 5 6 5 3]	2267.0	2	4	20000	2280.8	0.6
I5,141	75	[1 5 6 5 3]	2267.0	2	10	20000	2267.0	-0.0

What can also be observed is that the more difficult a problem becomes (in terms of problem size or objective function), the less competitive is solving the MILP model to obtain the exact optimal solution. This was to be expected due to the NP-hardness of the problem which makes the HFS scheduling problem intractable for exact algorithms.

Lower bounds used here were originally proposed for the identical machine case and adapted for unrelated machines by simply replacing all other machines by the fastest machine for a job at every stage. Therefore, they are expected to be weaker than for the case of identical machines. Nevertheless, MAJ−LS performs well, as the average distance of the results to the lower bound values is within an acceptable range.

A t-test for paired, dependent samples is conducted to statistically validate the performance of MAJ−LS compared to NEH. As a distribution-based test,

the t-test requires test data to be normally distributed. To test if a normal distribution can be assumed for MAJ−LS and NEH results, the chi-square test and Shapiro-Wilk test are performed. The outcome of those goodness-of-fit tests for normal distribution is shown in tables 6.3 and 6.4. Although normal distribution cannot be proven for all instances, the t-test is known to be robust as per the central limit theorem of statistics[16] for sample sizes > 30 (at a degree of freedom of 30, the Student's t-distribution is regarded as equalling the normal distribution). (Kim (2015))

Here, the null hypothesis H0, i.e. the mean of MAJ−LS results being equal to the mean of NEH results ($\mu_{MAJ-LS} = \mu_{NEH}$), is tested against the alternative hypothesis H2 that the mean of MAJ−LS results is smaller than the mean of NEH results ($\mu_{MAJ-LS} < \mu_{NEH}$). The results indicate that the null hypothesis can be rejected in all cases at the 0.05 significance level (see table 6.5), meaning that MAJ−LS outperforms NEH.

Table 6.3: Chi-square goodness-of-fit test for normal distribution on 192 instances per stage configuration.

Chi-square test for MAJ−LS and NEH results	**3 Stages**		**4 Stages**		**5 Stages**	
Hypothesis H0: Results are normally distributed	**Makespan**	**Flowtime**	**Makespan**	**Flowtime**	**Makespan**	**Flowtime**
significance level 0.01:						
H0 for MAJ−LS results	accepted	accepted	rejected	rejected	rejected	rejected
H0 for NEH results	accepted	rejected	rejected	rejected	rejected	rejected
significance level 0.05:						
H0 for MAJ−LS results	rejected	rejected	rejected	rejected	rejected	rejected
H0 for NEH results	rejected	rejected	rejected	rejected	rejected	rejected

Table 6.4: Shapiro-Wilk goodness-of-fit test for normal distribution on 192 instances per stage configuration.

Shapiro-Wilk test for MAJ−LS and NEH results	**3 Stages**		**4 Stages**		**5 Stages**	
Hypothesis H0: Results are normally distributed	**Makespan**	**Flowtime**	**Makespan**	**Flowtime**	**Makespan**	**Flowtime**
significance level 0.05:						
H0 for MAJ−LS results	rejected	rejected	rejected	rejected	rejected	rejected
H0 for NEH results	rejected	rejected	rejected	rejected	rejected	rejected

[16] The central limit theorem is the basis for parametric tests and states that a sample of sufficient size that is randomly selected can be used to estimate the parameters of the population using inferential statistics. If the sample size is sufficiently large, the means of samples obtained using a random sampling with replacement are distributed normally with the mean μ and the variance σ^2/n regardless of the population distribution. (see e.g. Kwak and Kim (2017))

Table 6.5: Dependent t-test for paired samples.

t-test **MAJ−LS vs. NEH**	3 Stages		4 Stages		5 Stages	
Alternative Hypothesis H2: $\mu_{MAJ-LS} < \mu_{NEH}$	**Makespan**	**Flowtime**	**Makespan**	**Flowtime**	**Makespan**	**Flowtime**
t-value	-9.3882	-13.9338	-8.0993	-13.5328	-8.2179	-13.8223
degrees of freedom	191	191	191	191	191	191
significance level	0.05	0.05	0.05	0.05	0.05	0.05
standard deviation of MAJ−LS − NEH	121.3233	67135.8	107.0398	80185.6	97.7428	81222.1
Hypothesis H0: $\mu_{MAJ-LS} = \mu_{NEH}$	rejected	rejected	rejected	rejected	rejected	rejected

To confirm the t-test outcome, the Wilcoxon test on a large sample using approximation[17] is conducted as the non-parametric equivalent to t-test without normal distribution requirement.

Again, the null hypothesis H0, $\mu_{MAJ-LS} = \mu_{NEH}$, is tested against the alternative hypothesis H2, $\mu_{MAJ-LS} < \mu_{NEH}$, and the null hypothesis can be rejected in all cases at the 0.05 significance level (see table 6.6). Therefore, MAJ−LS yields better results than NEH.

Table 6.6: Dependent Wilcoxon test for large samples.

Wilcoxon test MAJ−LS vs. NEH	3 Stages		4 Stages		5 Stages	
Alternative Hypothesis H2: $\mu_{MAJ-LS} < \mu_{NEH}$	**Makespan**	**Flowtime**	**Makespan**	**Flowtime**	**Makespan**	**Flowtime**
p-value	$3.85 \cdot 10^{-22}$	$1.48 \cdot 10^{-33}$	$2.21 \cdot 10^{-11}$	$1.48 \cdot 10^{-33}$	$5.62 \cdot 10^{-22}$	$1.48 \cdot 10^{-33}$
significance level	0.05	0.05	0.05	0.05	0.05	0.05
Hypothesis H0: $\mu_{MAJ-LS} = \mu_{NEH}$	rejected	rejected	rejected	rejected	rejected	rejected

An interim conclusion can be drawn with regards to the performance of MAJ−LS based on the results of the first optimization routine:

1. There seem to be two problematic cases of instance feature combinations. First, the processing time pattern "job correlation + diverse" (see table 5.1), which represents an unrealistic setting where processing times of jobs on machines vary drastically and do not follow any rule, often yields worse solutions compared to NEH (see table 6.7). This could be improved further by specific parameter adjustment.

   The second problematic case is that of "one-machine-only", where one or more stages contain only one machine. This problem leaves room for improvement (see table 6.8) and could be tackled by improving the pre-

[17] If the number of observations is such that $n(n+1)/2$ is large enough (> 20), a normal approximation can be used instead of exact probabilities for the signed rank sum test. (see e.g. Fellingham and Stoker (1964))

processor or some adjustments in the local search phase of the MAJ−LS heuristic.

2. Flowtime minimization seems to be more complicated to achieve than makespan minimization. Here, a 2−Opt improvement scheme can be optionally applied to the MAJ−LS solution if required.
3. The application of machine learning techniques can be useful as the results of MAJ−LS might be reproduced with only a fraction of the computation time by including machine learning predictions in the machine selection phase of the heuristic.

Table 6.7: Optimization routine results MAJ−LS without diverse cases.

MAJ−LS Results Without "Diverse" Instances	**3 Stages**		**4 Stages**		**5 Stages**	
Comparison	**Makespan**	**Flowtime**	**Makespan**	**Flowtime**	**Makespan**	**Flowtime**
MAJ−LS better than NEH	111	98	111	89	98	90
MAJ−LS same as NEH	44	0	0	0	44	0
MAJ−LS worse than NEH	13	70	57	79	26	78
Total no. of instances	168	168	168	168	168	168
Avg. deviation MAJ−LS to NEH (in %)	5.58	1.60	4.75	1.40	4.16	1.80
Std. dev. of deviation	7.47	3.05	6.75	3.16	6.11	3.57
Maximum improvement by MAJ−LS to NEH (in %)	37.90	14.85	38.47	13.91	38.97	23.35
Maximum deterioration by MAJ−LS to NEH (in %)	-1.46	-1.74	-2.93	-2.37	-3.35	-1.99

Table 6.8: Optimization routine results MAJ−LS without one-machine cases.

MAJ−LS Results Without "One-Machine" Instances	**3 Stages**		**4 Stages**		**5 Stages**	
Comparison	**Makespan**	**Flowtime**	**Makespan**	**Flowtime**	**Makespan**	**Flowtime**
MAJ−LS better than NEH	95	91	83	79	85	87
MAJ−LS same as NEH	0	0	0	0	0	0
MAJ−LS worse than NEH	7	11	5	9	6	4
Total no. of instances	102	102	88	88	91	91
Avg. deviation MAJ−LS to NEH (in %)	9.08	2.96	8.70	3.22	7.71	4.02
Std. dev. of deviation	7.73	3.21	6.97	3.39	6.43	3.56
Maximum improvement by MAJ−LS to NEH (in %)	37.90	14.85	38.47	13.91	38.97	23.35
Maximum deterioration by MAJ−LS to NEH (in %)	-1.46	-0.81	-1.47	-1.11	-3.35	-1.66

In a next optimization run, the additional improvement step based on 2−Opt algorithm (see chapter 5.2.2) was included for instances where the flowtime of MAJ−LS was higher than that of NEH. Algorithm 2−Opt improves the performance of MAJ−LS, if required, and for the majority of cases the flowtime is lower than that of NEH, whereas the remaining solutions are very close to NEH (see table 6.9).

Table 6.9: Improvement of flowtime solutions by 2-Opt.

2-Opt-Improvement for MAJ-LS Results with Higher Flowtime Values than NEH	3 Stages	4 Stages	5 Stages
No. of instances (NEH flowtime <MAJ-LS flowtime) before 2-Opt	86	99	98
Avg. deviation (NEH flowtime < MAJ-LS flowtime) before 2-Opt in %	-0.87	-1.40	-1.50
No. of instances (NEH flowtime < MAJ-LS flowtime) after 2-Opt	19	29	21
Avg. deviation (total) after 2-Opt -improvement in %	0.58	0.76	0.80
Avg. deviation (NEH flowtime < MAJ-LS flowtime) after 2-Opt for results still worse than NEH in %	-0.04	-0.08	-0.08

Summarizing, the first optimization routine yielded promising results. A cross section of the proposed testbed is selected for further investigations. The selected data instances are presented in table 6.10 and cover all possible characteristic values of the testbed features described in section 5.1.2.

Table 6.10: Cross section of data instances for further evaluation.

Instance	Machine Configuration	No. of Jobs	Machines per Stage	Job Correlation	Machine Correlation
3 Stages, 1	constant	25	[2 2 2]	equal	-
3 Stages, 20	equal	50	[2 2 2]	equal	-
3 Stages, 95	ascending	25	[3 4 6]	hill	-
3 Stages, 165	hill	100	[3 5 3]	-	equal
4 Stages, 36	descending	50	[6 5 3 3]	job families	-
4 Stages, 68	equal	100	[2 3 3 3]	job families	-
4 Stages, 100	valley	25	[5 2 3 4]	-	valley
4 Stages, 112	valley	50	[4 2 3 4]	-	ascending
4 Stages, 128	equal	50	[3 2 1 3]	-	valley
5 Stages, 33	hill	50	[3 5 6 5 3]	job families	-
5 Stages, 37	constant	75	[2 2 2 2 2]	-	equal
5 Stages, 59	ascending	100	[3 3 4 5 7]	equal	-
5 Stages, 150	descending	75	[7 5 4 3 3]	-	descending

6.2 MAJ-ML Heuristic

The results of MAJ-LS can be reproduced with only a fraction of the original computation time by including ML predictions in the machine selection phase of the heuristic, as described in section 5.3.1. A preliminary study confirmed that similar solutions can be achieved in less time as is shown for different problem sizes and parameter settings in figures 6.1 and 6.2. Obviously, including ML predictions is only advantageous if time is a constraint and if the behaviour of the underlying production process is known and controllable, so that no new training is required for new planning horizons.

Further experiments are conducted on selected instances presented in table 6.10. First, training of NN and SVM is performed in EIDOminer with training data generated by algorithm 7 (HFS_LernDataGen) for 20 randomly

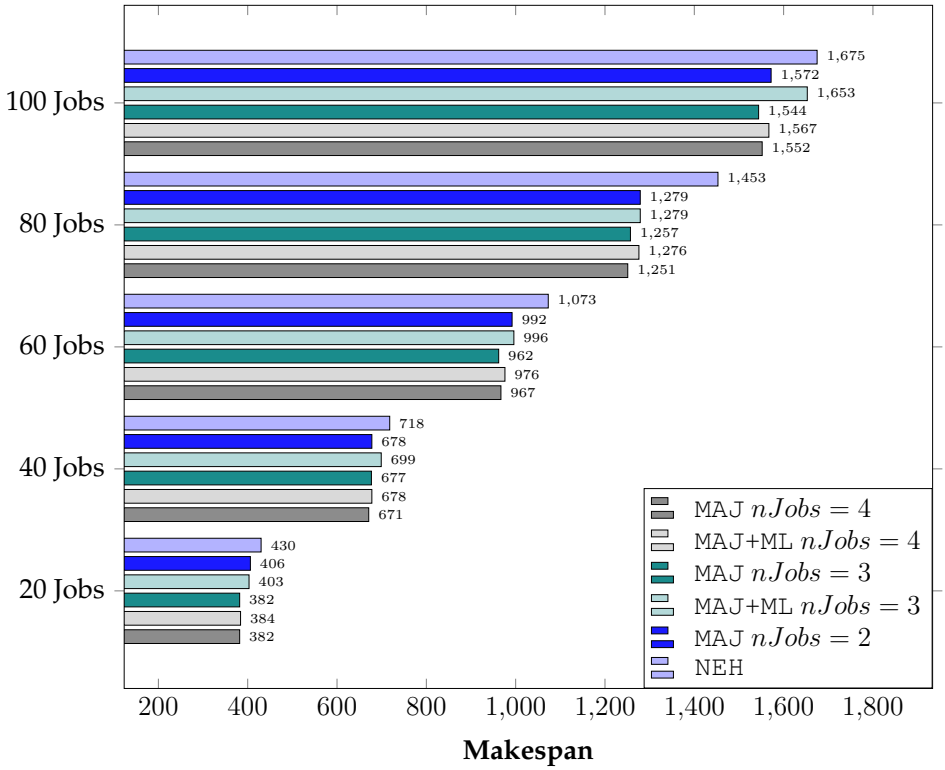


Figure 6.1: Comparison of makespan values of MAJ, MAJ−ML, and NEH.

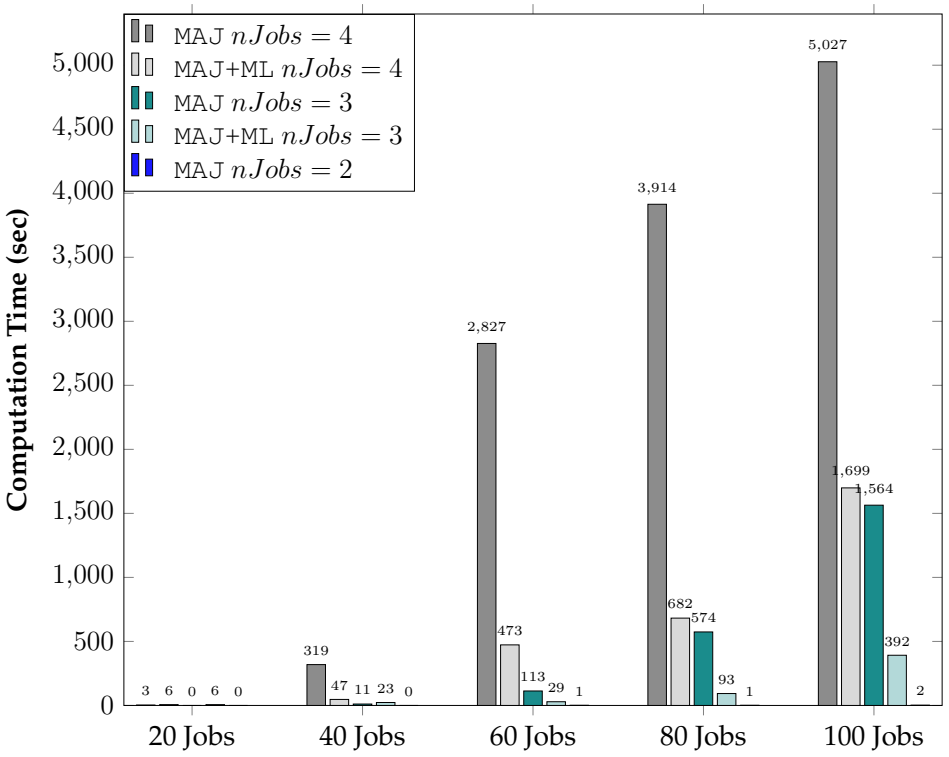


Figure 6.2: Comparison of computation times of MAJ and MAJ−ML.

selected jobs of each instance. Here, automated hyperparameter optimization of ML algorithms and integrated fusion methods are utilized, but no further optimization of the ML prediction function is performed. Then, a schedule for the same instance is generated by algorithm 8 (`HFS_Prediction`) and compared with the initial `MAJ` solution of algorithm 1 (`HFS_Scheduler`). The main objective if this first experiment is to get an idea of possible time savings by utilizing ML techniques.

An overview of the results is given in table 6.11. As it is the case with `MAJ`, the hybrid approach also provides an option for setting parameters in the training and prediction phase. First, training data can be generated with the complete list or just a subset of all jobs and for different `nJobs`-values. Then, prediction quality of the trained ML model can be influenced during the training phase by various means, as described in section 5.3.2. Finally, algorithm 8 (`HFS_Prediction`) can be controlled by the parameters `addStages` and `addJobPackages`.

Table 6.11: Comparison of `MAJ` and `MAJ+ML` results for selected instances.

Selected Problems			MAJ Results				MAJ+ML Results							
Instance	No. of Jobs	Machines per Stage	Makespan	Comp. Time [sec] Makespan	Flowtime	Comp. Time [sec] Flowtime	Makespan	Deviation [%]	Comp. Time [sec] Makespan	Deviation [%]	Flowtime	Deviation [%]	Comp. Time [sec] Flowtime	Deviation [%]
3 Stages, 1	25	[2 2 2]	443	59	6256	29	447	1.0	6	-90.6	6710	7.3	20	-32.3
3 Stages, 20	50	[2 2 2]	798	894	21115	2	796	-0.3	427	-52.2	21224	0.5	3	50.9
3 Stages, 95	25	[3 4 6]	486	62	6884	45	544	12.0	51	-17.2	7074	2.8	44	-2.2
3 Stages, 165	100	[3 5 3]	1584	12280	77203	1438	1676	5.8	1752	-85.7	80762	4.6	301	-79.0
4 Stages, 36	50	[6 5 3 3]	1054	88	21542	68	1071	1.6	57	-35.6	21583	0.2	47	-30.4
4 Stages, 68	100	[2 3 3 3]	2548	4529	109031	4759	2755	8.1	104	-97.7	111315	2.1	105	-97.8
4 Stages, 100	25	[5 2 3 4]	577	11	8874	9	610	5.8	12	9.9	9939	12.0	12	37.3
4 Stages, 112	50	[4 2 3 4]	993	108	26623	103	1088	9.6	42	-61.4	28057	5.4	43	-57.9
4 Stages, 128	50	[4 2 1 3]	1652	44	43895	40	1662	0.6	21	-51.1	46440	5.8	18	-54.9
5 Stages, 33	50	[3 5 6 5 3]	1139	15	25420	190	1293	13.5	6	-62.5	26305	3.5	101	-46.7
5 Stages, 37	75	[2 2 2 2 2]	1245	15	50930	15	1266	1.7	8	-48.0	52428	2.9	8	-48.3
5 Stages, 59	100	[3 3 4 5 7]	1074	9581	58012	9184	1086	1.2	716	-92.5	61178	5.5	606	-93.4
5 Stages, 150	75	[7 5 4 3 3]	1227	1676	51255	1549	1389	13.2	253	-84.9	60647	18.3	170	-89.0

The results indicate that, although a time reduction with moderate increase of the objective function value can be achieved in most cases, the application of ML is particularly suitable for larger instances, as is shown with regards to the number of jobs to be scheduled in table 6.12. A very satisfactory result is that instances with initially high computation times also yield very high time savings by including ML without a major loss of solution quality. For example, considering instances with 100 jobs, 92% [90%] of computation time can be saved in average while makespan [flowtime] increases by 5% [4%] correspondingly.

Table 6.12: Comparison of MAJ+ML results for different instance sizes.

No. of Jobs	Avg. Deviation of Makespan [%]	Avg. Deviation of Comp. Time [%] Makespan	Avg. Deviation of Flowtime [%]	Avg. Deviation of Comp. Time [%] Flowtime
25	6.2	-32.6	7.3	0.9
50	5.0	-52.6	3.1	-27.8
75	7.5	-66.5	10.6	-68.7
100	5.0	-92.0	4.1	-90.1
Average	**5.7**	**-59.2**	**5.4**	**-41.8**

Until now, MAJ+ML was applied to the same instances that were used to generate training data. For each training instance, a subset of 20 jobs was randomly selected and scheduled by MAJ. This schedule was then used for training and, based on the respective trained ML model, a complete schedule was determined by MAJ+ML. For comparison purpose, the complete instance including all jobs was then additionally solved by MAJ.

In a next step, MAJ+ML is applied to randomly generated new instances. These instances follow the same feature combination as the initial training instance with regards to the structure explained in chapter 5.1.2, e.g. they have the same processing time pattern, but the total number of jobs to be scheduled varies between 50, 100 and 300.

One example each for makespan and flowtime minimization is selected to generate new test instances and to compare the solution to that of NEH. Furthermore, the influence of the parameter *nJobPackages* of algorithm 8 (HFS_Prediction) on the solution quality and computation time is investigated. As mentioned earlier, ML tools are able to predict the objective function values for different job packages with high accuracy and the prediction function can be further optimized by selecting the optimal fusion method as is shown by error measures of different fusion methods for the example instances in table 6.13.

As can be seen in tables 6.14 and 6.15, the hybrid approach is competitive in comparison to NEH and by setting the parameter *nJobPackages*, the solution quality can be influenced. Again, the known trade-off between quality and computation time occurs. If even further improvement is required, there

Table 6.13: RRMSE values of different fusion methods of the selected trained models for extrapolation.

Training Instance	ML Setting	Majority Voting RRMSE	Average RRMSE	Weighted Average RRMSE	PCA MR RRMSE
4 Stages no. 112, 50 Jobs, Valley Machine Order, Machine Correlation, Ascending Processing Times, Makespan Minimization	NN and SVM with 10-fold cross validation	0.0114	0.0196	0.0090	0.0052
4 Stages no. 36, 50 Jobs, Descending Machine Order, Job Correlation, Job Families, Flowtime Minimization	NN and SVM with 10-fold cross validation	0.0062	0.0086	0.0070	0.0083

are several ways to do so. First, training data can be optimized by further adjustment of MAJ parameters and the accuracy of the prediction function applied in MAJ+ML can be increased by more training cycles and hyperparameter optimization of ML tools. Furthermore, the local search improvement steps (algorithms 5 and 6) can be easily applied on the MAJ+ML solutions as well.

Table 6.14: Comparison of makespan results provided by MAJ+ML for different instance sizes and parameters.

No. of Jobs	Parameter $nJobPackage$	Avg. Makespan **MAJ+ML**	Avg. Computation Time [sec]	Avg. Makespan **NEH**	Avg. Difference of Makespans [%]
50	2	1105.3	24.6	1134.5	2.0
100	2	2023.8	84.3	1981.9	-2.5
300	2	5578.1	1500.6	5359.7	-4.2
50	20	1049.4	37.0	1134.5	6.8
100	20	1961.5	113.8	1981.9	0.6
300	20	5387.7	2181.0	5359.7	-0.7
50	100	1032.1	87.6	1134.5	8.2
100	100	1926.2	237.4	1981.9	2.4
300	100	5325.2	5019.2	5359.7	0.5

Table 6.15: Comparison of flowtime results provided by MAJ+ML for different instance sizes and parameters.

No. of Jobs	Parameter $nJobPackage$	Avg. Flowtime **MAJ+ML**	Avg. Computation Time [sec]	Avg. Flowtime **NEH**	Avg. Difference of Flowtime [%]
50	2	22796.5	32.5	24256.6	5.9
100	2	76470.7	128.5	81971.6	6.7
300	2	605165.6	2204.3	663137.7	8.7
50	20	22404.9	48.7	24256.6	7.6
100	20	75681.4	177.5	81971.6	7.6
300	20	602036.6	2842.5	663137.7	9.2
50	100	22203.2	81.7	24256.6	8.4
100	100	76001.0	284.8	81971.6	7.2
300	100	601469.8	6026.3	663137.7	9.3

6.3 Stability of the MAJ Approach

For practical applications not only computation time and quality of results are important, but also their consistency over time. This means that the stability of different algorithms has to be considered and compared. In numerical analysis, a numerically stable algorithm avoids magnifying small errors (see e.g. Hull and Luxemburg (1960)). With regards to scheduling algorithms, small changes in the input data, here e.g. job processing times, should not lead to drastic changes in the objective function value.

To compare the stability of the MAJ approach and NEH, two different tests are carried out. First, we evaluate the results of the examples presented in chapter 6.2, where new instances for a given feature combination are repeatedly generated and solved, with respect to the variation of those results. In a second step, we assume that the job sequence and machine assignment once determined by MAJ approach and NEH are kept for next planning horizons. This approach could be suitable for stable production processes, i.e. processes with small variations in process data and order volumes amongst planning periods. Here, two settings are tested. In the first test, the mean and standard deviation of processing times are kept constant while in the second test, noise is additionally introduced by adding randomly generated time surcharges to the normally distributed processing times.

In this work, solution stability is measured by the range and distribution of the results provided by MAJ and NEH. For the first comparison, box plot diagrams[18] for different problem sizes and parameter settings are presented in figures 6.3 to 6.8.

[18] Box plots graphically depict quartiles of the underlying data. The box represents the first to third quartile, with a horizontal line indicating the second quartile (median). The whiskers show the most extreme points not considered outliers, which are are displayed using "+". (Behrens (1997))

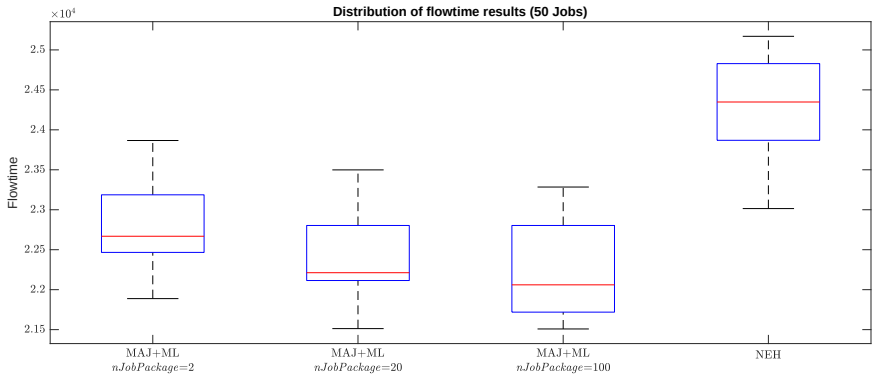


Figure 6.3: Box plot diagram of MAJ+ML and NEH flowtime results at the example of 50 jobs.

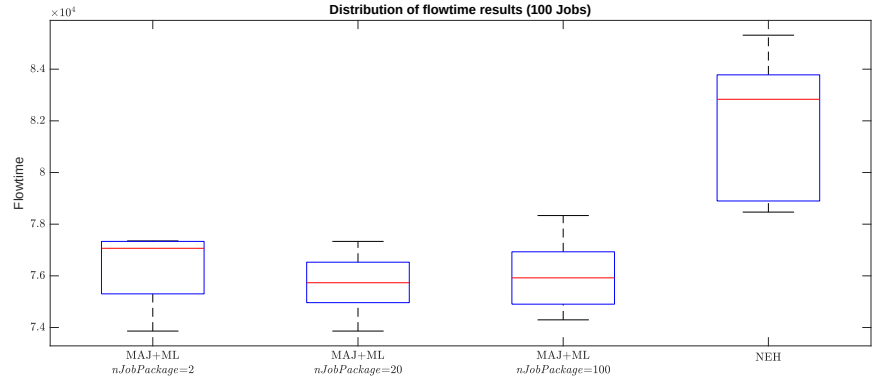


Figure 6.4: Box plot diagram of MAJ+ML and NEH flowtime results at the example of 100 jobs.

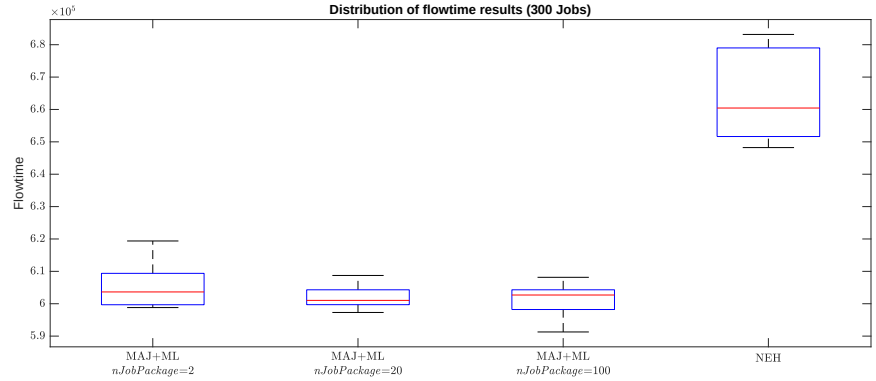


Figure 6.5: Box plot diagram of MAJ+ML and NEH flowtime results at the example of 300 jobs.

Several conclusions can be drawn from the presented examples. First, MAJ+ML produces better objective function values than NEH on average with higher improvements in the flowtime case. Secondly, the parameter $nJobPackage$ seems to have a more crucial influence in the case of makespan minimization and generally, increasing $nJobPackage$ yields better solution quality. Furthermore, MAJ+ML results are more *consistent*, as the range of the solutions for similar input data is smaller than that of the NEH results. Overall, the MAJ+ML approach seems to produce more stable results than NEH and can be adjusted by parameter selection to suit the application.

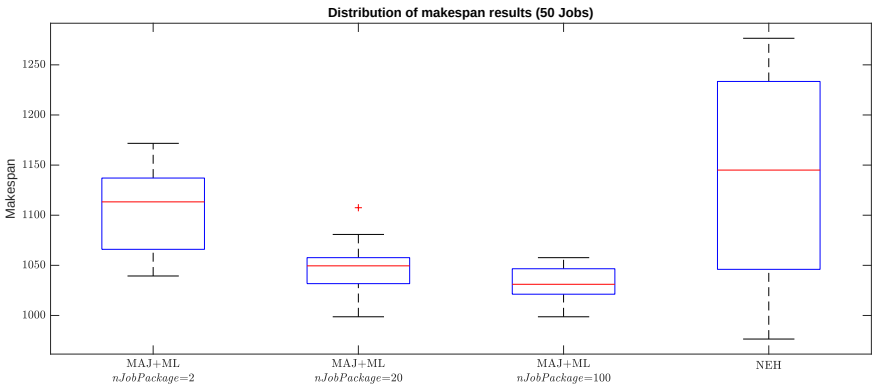


Figure 6.6: Box plot diagram of MAJ+ML and NEH makespan results at the example of 50 jobs.

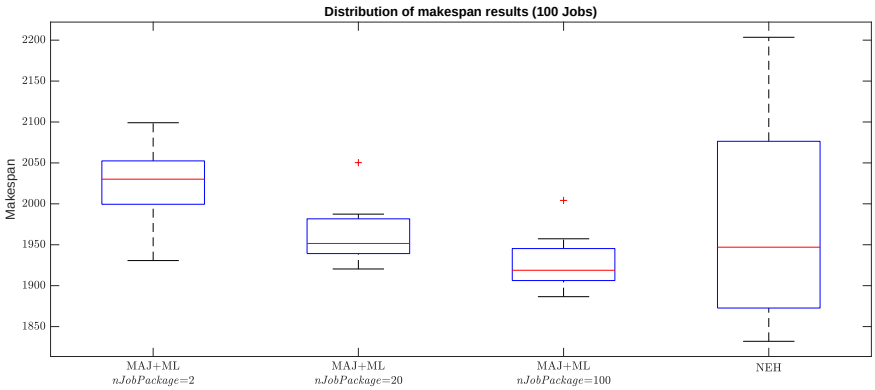


Figure 6.7: Box plot diagram of MAJ+ML and NEH makespan results at the example of 100 jobs.

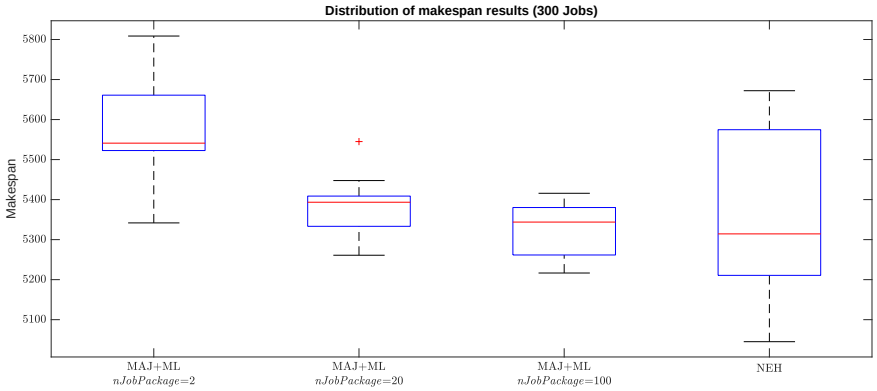


Figure 6.8: Box plot diagram of MAJ+ML and NEH makespan results at the example of 300 jobs.

For the first test of the second stability experiment, 500 new instances are generated according to the examples presented in 6.2. For both example instances, the original MAJ+ML and NEH schedules are taken as basis for scheduling the new instances, i.e. the job sequence and machine assignment once determined are kept. This means that both algorithms are only run one time and afterwards, only the new corresponding makespan and flowtime values have to be calculated based on the already known optimized machine assignment and sequence provided by MAJ and NEH. Again, box plot diagrams are used to compare the results as shown in figures 6.9 and 6.10. Following this approach, MAJ+ML still leads to better results than NEH with regards to solution stability, but the range of results is generally reduced in both cases. As can be seen in both diagrams, MAJ+ML is clearly advantageous compared to NEH in terms of solution quality.

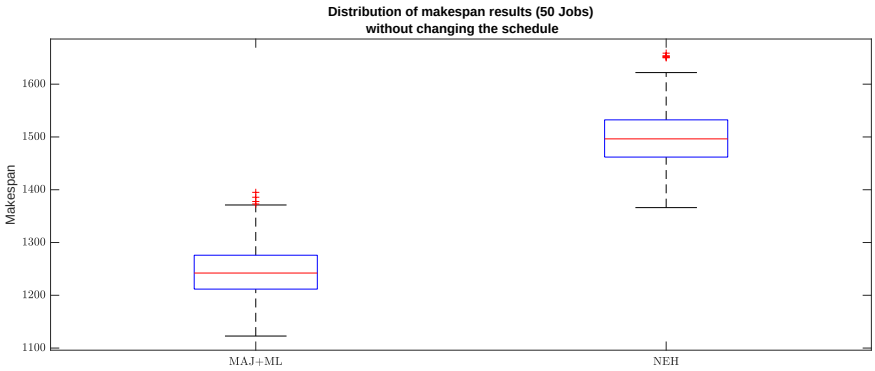


Figure 6.9: Box plot diagram of MAJ+ML and NEH makespan results for a fixed solution at the example of 50 jobs and 500 runs.

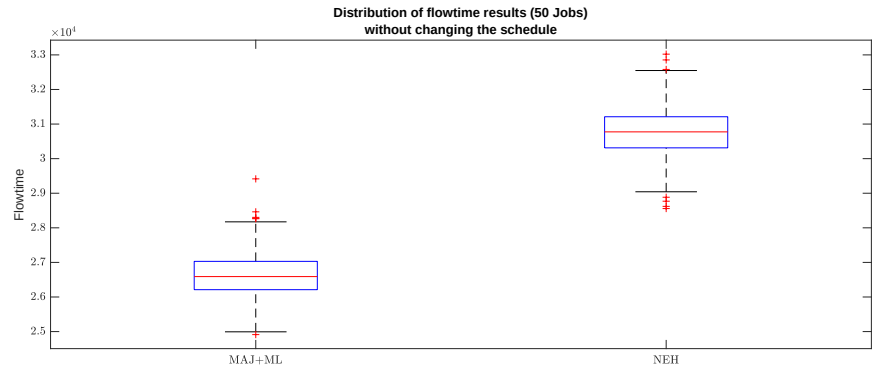


Figure 6.10: Box plot diagram of MAJ+LS and NEH flowtime results for a fixed solution at the example of 50 jobs and 500 runs.

The second test of the experiment follows the same approach, but initial solutions are determined by MAJ and noise is added to the newly generated test instances. Here, 3000 runs are executed and the results are presented in figures 6.11 and 6.12. A similar outcome can be detected as to the case without adding noise. Again, MAJ generates more stable solutions with much smaller makespan and flowtime values than NEH. As can be seen in table 6.16, the standard deviation of NEH results is significantly higher than that of MAJ results.

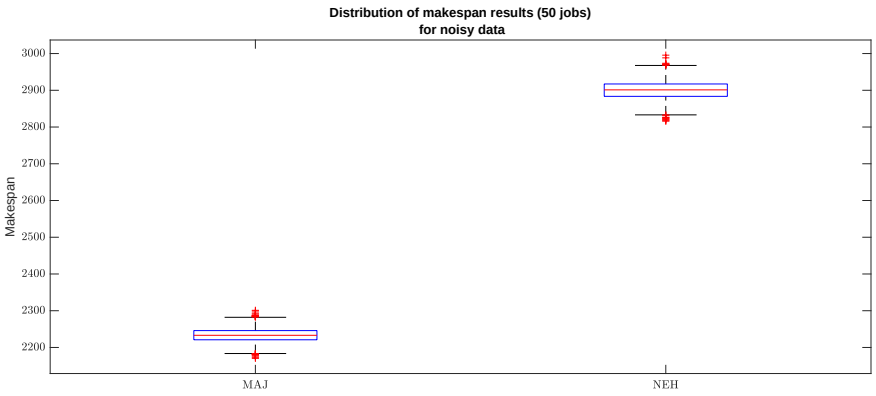


Figure 6.11: Box plot diagram of MAJ and NEH makespan results for a fixed solution at the example of 50 jobs and 3000 runs.

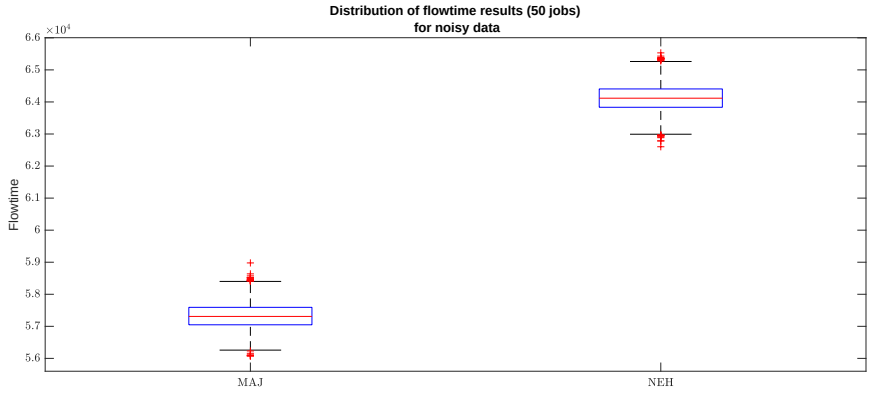


Figure 6.12: Box plot diagram of MAJ and NEH flowtime results for a fixed solution at the example of 50 jobs and 3000 runs.

Table 6.16: Statistical measures of MAJ and NEH results for a fixed solution at the example of 50 jobs and 3000 runs including noise.

Measure	MAJ Makespan	NEH Makespan	MAJ Flowtime	NEH Flowtime
Minimum	2170	2816	56072	62602
Maximum	2301	2996	58978	65531
Range	131	180	2906	2929
Mean	2233	2901	57320	64123
Median	2233	2901	57309	64117
Standard Deviation	18.23	24.26	410.64	427.80

7 Conclusion

This work proposed a new approach to hybrid flow shop scheduling with unrelated machines, consisting of two variants of scheduling algorithms, for the minimization of either makespan or flowtime. A comprehensive literature review illustrated the necessity of generating effective and efficient heuristics that are able to deal with the special case of unrelated machines as well as with different stage, machine, and job configurations of the hybrid flow shop.
In addition, a representative testbed based on the work of Watson et al. (2002) was introduced to evaluate the performance of the approach with test data aiming at realistic instances representing real-world production processes. Furthermore, benchmark results were generated to validate the performance. New lower bounds for makespan minimization were proposed. To generate upper bounds for makespan and flowtime optimization, a modified version of the well-known `NEH` heuristic was used, as well as an MIP model that was solved with a time restriction by a commercial solver.
In the first variant of the approach, called `MAJ`, a new heuristic based on *divide et impera* was presented that allows the decision maker to adjust different parameters, which influence the decomposition steps in several ways, and thereby enable a predefined solution quality to be achieved. A pre-processor may dismiss some unreasonable options of machine assignments in advance. In addition, several post-processing local search improvement schemes can be optionally performed, which is indicated in the result section as `MAJ-LS`.
In the second variant, namely `MAJ+ML`, machine learning techniques were utilized to accelerate the computation. SVM and NN combined with different fusion methods are applied to predict the optimized objective function values for all possible job packages of a predefined size, so that corresponding machine assignments have to be derived only for the packages selected for the ultimate solution. Including ML techniques in this way yields an average reduction of computation time of up to 92%. In this context, the extensive literature review showed that there are many works aiming at combining ML techniques

with traditional approaches to combinatorial optimization and that the discrete nature of such problems seems to be challenging for pure ML techniques. Therefore, hybrid approaches are gaining popularity lately. Very few authors employing ML in scheduling though consider hybrid flow shops and, to the best of our knowledge, none has combined ML and heuristics in the way presented in this work.

The experimental evaluation of the proposed approach was threefold. First, the MAJ variant was benchmarked against the lower and upper bounds, including statistically validating the performance of MAJ. Then, MAJ+ML was tested with emphasis on computational time. Finally, the stability of the new approach was compared to that of NEH.

Overall it can be summarized that the new approach presented here outperforms the existing methods with regards to the achieved objective function values and solution stability. Including ML techniques proved to be successful in drastically reducing computation time while maintaining good makespan and flowtime solutions. A few details of the results shall be highlighted in the following.

In contrast to NEH heuristic, which only generates a single solution, MAJ provides the option to deliberately influence solution quality and time. The trade-off between higher computation time and lower objective function values can be actively controlled by the decision maker. This can be done by activating the described pre- and several post-processing improvement schemes and by means of different core parameters of the MAJ heuristic. In addition, ML tools can be included if computation time is a constraint and the time savings achieved generally justify the small deterioration of makespan and flowtime results.

The presented results can be interpreted as an initial test, and plenty of room exists for improvement just by adjusting the parameters individually for each test instance. In this work, we used a few standard configurations to gather information on the overall performance of the new approach as well as on how different test instances reflect in the solutions. Nevertheless, the superior per-

formance of MAJ could be statistically validated.

Looking at the stability of the algorithms, MAJ provides far more stable solutions with respect to the range and distribution of results than NEH when solving multiple instances of same input data categories as well as for noisy data. This conclusion holds for both objective functions considered in these experiments.

8 Outlook

This work proposed a new generic approach to makespan and flowtime optimization in hybrid flow shops and presented a new method to accelerate computation times through machine learning techniques. The proposed heuristics showed promising results with respect to solution quality and stability, and provide ample scope for future research.

Regarding the new heuristic, the pre-processor can be further optimized to narrow the solution space before running the main optimization routine. In addition, more post-processor improvement schemes can be tested. In this work, the different parameters were set consistently and were not optimized for the different instance categories. Here, future work may explore the relationship between input instance and optimal parameter settings, as it was shown that different production settings require different parameter configurations. Such a relationship might be discovered e.g. by design of experiments or by data mining techniques and machine learning.

From a user perspective, the GUI can be extended so that an estimation of objective function values and computation times for different algorithms and parameter settings is provided. Based on those estimations, the user can then choose a preferred option and automatically run the scheduler.

Furthermore, the `MAJ` approach can be applied in practical applications, and production constraints as well as additional objectives can be included in the scheduling model. Also, it might be used in simulation studies to define improved machine layouts when a modification of the process layout is considered.

Aiming at the integration of machine learning in scheduling approaches, several methods may be promising fields of future work. First, the ML models used in this work might be enhanced by other tools, e.g. Bayesian networks or decision trees, and the hyperparameters of the ML tools can be optimized. Furthermore, the proposed ML model can be utilized in other metaheuristics in a similar way, e.g. to estimate fitness functions or expected objective function values of different solution candidates.

Other applications of ML techniques might be explored to focus on how optimized machine assignments or sequences might be predicted directly by ML models. From the experience we made and based on the literature, this might be interesting but challenging, due to the discrete nature of the optimization problem and the difficulty of generating 'optimal' training data for NP hard problems.

Finally, no valid lower bounds for the flowtime in HFS scheduling problems could be found, so that it might be worth looking at ways to develop such lower bounds.

Appendix

Table A.1: Machine learning applications in scheduling literature.

Author and Year	Key Aspects	ML Algorithm	Optimization Problem	Objective Category	Prediction
Hopfield and Tank (1985)	optimization with NN	recurrent NN	TSP	distance	binary permutation matrix
The proposed HNN is a fully connected recurrent artificial neural network with only one layer. Neurons represent a binary permutation matrix where each decision variable represents a position of a city in the route. Typically, the network energy function includes the objective function of the optimization problem which needs to be minimised, and the constraints of the problem which are included as penalty terms. The energy function is also required to be a Liapunov function, so that consecutive updates of the Hopfield network constantly decrease the system energy until a stable local minimum is located. Different learning rules exist to minimize the energy function by repeatedly (synchronously or asynchronously) updating neurons' states. Limitations can be related to complexity, computation time and local minima results.					
Simon et al. (1988b)	stochastic HNN, SA	recurrent NN	job shop	flowtime	pairwise precedences
The authors propose a stochastic HNN, integrating a Boltzmann machine, and the Job Shop problem is mapped into a 2- dimensional matrix representation of neurons similar to TSP structure. Biases enforce the operation precedence relationships. The solution to the Job Shop problem is represented by a set of cost function trees encoded in the neuron matrix where nodes in the set of trees represent a job, and links represent interdependencies between jobs. The cost attached to each link is a function of the processing time of a particular job. The starting time of each job can be determined by tree structure. Using a simulated annealing algorithm, the temperature of the system is slowly decreased according to an annealing schedule until the energy of the system is at a local or global minimum. The authors point out that for large size problems, this approach is not feasible.					
Simon et al. (1988c)	stochastic HNN, SA	recurrent NN	job shop	flowtime	pairwise precedences
Second part of same study.					
Simon et al. (1988a)	integer linear programming HNN	recurrent NN	job shop	flowtime	pairwise precedences
In this paper, an integer linear programming neural network based on a modified Tank and Hopfield neural network model is proposed. Biases enforce the operation precedence relationships. The solution to the Job Shop problem is represented by a set of cost function trees encoded in the neuron matrix where nodes in the set of trees represent a job, and links represent interdependencies between jobs. The cost attached to each link is a function of the processing time of a particular job. The starting time of each job can be determined by tree structure. Constraints of the Job Shop problem are formulated as a set of integer linear equations that are solved by an iterative linear programming with integer adjustments technique,where the linear and nonlinear zero-one variables are represented by linear sigmoid and nonlinear high-gain amplifiers with a response of a step function. Authors state that the proposed approach shows considerable advantages over Hopfield TSP-type network in terms of total number of interconnections and operational amplifiers but others mention that it generates constraint-violating solutions					
Lo and Bavarian (1991)	parallel machines, 3D-HNN	recurrent NN	parallel unrelated machines	makespan	binary: assignment of jobs to machine at time intervals
The HNN is extend to a 3-dimensional binary neuro-box network. The z-axis represents the jobs, the x-axis represents the machines and the y-axis represents the time. Neurons represent an ordered list of jobs on the machines. Connections between the neurons are not symmetric as in the HNN, so the global convergence is not guaranteed, however convergence to any sub-optimal valid solution is also acceptable for practical purposes. Energy function consists of two parts: the physical objective function and the constraint functions. The authors state that the neuro-box network concept allows expansion in any of the three dimensions, i.e., number of jobs, number of machines and planning period but large size problems are not feasible.					

Zhou et al. (1991)	linear energy function HNN	recurrent NN	job shop	sum of starting times of jobs' last operation	operations starting times
A HNN with linear energy function is proposed where neurons represent starting times of operations of a job and the energy function consists of job starting times and penalty terms for constraint violations. The authors emphasis the scaling advantage (due to less number of neurons and nodes) compared to traditional HNN and integer linear programming NN.					
Nakasuka and Yoshida (1992)	dynamic scheduling, DR, DT, knowledge base	DT	flow shop	makespan, constraint: tardiness limit	DR selection
A Binary decision tree for fixed scheduling intervals where a best dispatching rule is chosen based on the current system's parameters, is proposed. Results of a simulation study and attribute extraction are used as input for inductive learning, also called concept learning. In this algorithm, a binary decision tree is automatically generated using empirical data obtained by iterative production line simulations, and it decides in real time which rule to be used at decision points during the actual production operations.					
Arizono et al. (1992)	stochstic NN, Gaussian machine, JIT, setup times	recurrent NN	single machine	flowtime	binary permutation matrix
The proposed Gaussian machine modification represents a discrete-time and asynchronous-transition mode HNN. Neurons represent binary variables assigning jobs to positions of the sequence. The energy function contains the total actual flowtime and penalty terms for constraint violations. Gaussian approach uses annealing and sharpening schemes to enhance convergence. The authors claim that, whenever the energy function of the network is defined based on the constraints and the performance measure, the network can autonomically converge to the minimum energy state based on the simple system dynamics.					
Shaw et al. (1992)	pattern-directed inductive learning, knowledge base	DT	FMS	tardiness	DR selection
In this study, a decision tree classifies manufacturing patterns based on system attributes and provides a corresponding dispatching rule selection for each pattern. The inductive learning approach uses simulation study for training example generation and afterwards includes a rule refinement and pattern smoothing module. The pattern-directed method enables the scheduler to classify distinct manufacturing patterns and to generate a decision tree consisting of heuristic policies for dynamically selecting the dispatching rule appropriate for a given set of system attributes.					
Hellstrom and Kanal (1992)	mean-field theory NN, asymmetric NN, energy-gradient decomposition	recurrent NN	parallel identical machines	tardiness	assignment of jobs to machines
The partitioning problem formulation for the knapsack problem is used and represented by an analog, non-neural network: Instead of probabilistically adopting a state of 0 or 1, the units adopt states that model the expected values of analogous units (meanfield model) . In the parallel decomposition, each resulting process attains neuron-like behaviour, that is, continuous integration and graded thresholding. A modular NN is proposed consisting of clusters of four hidden units and one visible unit. Neurons represent a binary matrix (assignment of tasks to processors) combined with task size. A program-constructive approach is followed: A series of stability-preserving program transformations is applied to a model describing the non-neuronlike behaviour of a collection of binary-state processing elements representing the scheduling objective and constraints. The resulting transformed programs correspond to non-neuronlike analog, and analog neural networks. A program-constructive approach to deriving asymmetric mean-field networks for multiprocessor scheduling avoids the problem of 0-I Hamiltonian reducibility by defining hidden units which reconstruct arbitrary energy gradients either by piecewise composition or by approximation with a series of graded thresholds. The authors state that this approach has several advantages over the existing HNN formulations as per the authors. In particular the requirements for functionally homogeneous visible units and 0-1 Hamiltonian reducibility of the objective function are circumvented. The process of creating a neural network from an arbitrary discrete optimization problem is broken into a sequence of manageable steps.					
Vaithyanathan and Ignizio (1992)	resource constrained scheduling problem, knapsack models, stochastic HNN	recurrent NN	resource constrained scheduling	schedule as many jobs as possible without violating the resource constraints	assignment of jobs to machines

A resource constrained scheduling problem is first decomposed into a series of multidimensional knapsack models and an equivalent neural network model for this particular representation is established. A new stochastic neural network is proposed as modeled upon extensions to HNN: The proposed network includes "external" neurons which serve to provide stochasticity in the search process. The first layer equals a HNN where neurons represent binary variable for jobs assigned to resources, the output of the second layer is used to indicate those jobs that have been scheduled (n neurons). The output of the third layer indicates just how effectively each available resource has been used (m neurons). An iterative approach is followed: The upper limit on possible solutions is evaluated, a stability check is performed and transient feedback is employed. The authors state that the stochastic element shall avoid local minima solutions.					
Burke and Ignizio (1992)	survey	NN	OR problems	survey	survey
Emphasis on Operations Research problem classes, applicability and limitations of different forms of neural networks.					
Johnston and Adorf (1992)	complex constraints, dynamic and stochastic scheduling, long-term scheduling	recurrent NN	NASA/ESA Hubble Space Telescope, long-range astronomical observations	quality of obtained data, impact of unpredictable factors in schedule, observation priority	assignment of tasks to time intervals
A discrete stochastic network instead of a continuous deterministic one using a directed weighted constraint graph representation of the scheduling problem is proposed as an alternative to the Boltzmann machine. It is a general, parallelizable, randomized, heuristic neural network algorithm suitable for a variety of constraint satisfaction and combinatorial optimization problems. The basic network without stopping rule can be considered as a stochastic multi-start algorithm of the Las Vegas type: It either finds a solution or, with some low probability, announces to have failed to find an answer. If the network finds a solution, it is always a feasible. Neurons represent binary decision variables assigning tasks to time intervals. No network training is required: Instead, the network is designed entirely and automatically from the scheduling constraints, both strict and preference. The approach integrates a quantitative representation of "hard" constraints and "soft" preferences which exploits evidential reasoning techniques and which can be automatically translated into the biases and connection weights of a neural network, a network topology using multiple asymmetrically-coupled networks with different time constants to represent and enforce certain types of strict scheduling constraints, and a network dynamics consisting of discrete, deterministic neurons with a stochastic selection rule.					
Liang et al. (1992)	Markov optimization for data preprocessing, circuit-board manufacturing process	BPN	FMS	throughput, product quality	DR selection
A BPN is used for real-time scheduling in a circuit board manufacturing process. The input layer represents system state and the output layer selects the best scheduling decision. The corresponding decision simulation for generating training data was performed by different human experts (more than one hundred undergraduate industrial engineering student subjects from a Big Ten university). A semi-Markov decision model can be applied to remove inconsistent and erroneous data. The purified training data is then used to build ANN models that exploit the expertise more advantageously. The authors state that there are two major advantages for integrating semi-Markov processes with ANN. First, the redundant and inconsistent data can be removed from the training data. This increases the reliability of the resulting model. Second, because the optimization process results in one single optimal policy for each state, the size of the training data can be reduced dramatically.					
Looi (1992)	survey	NN	combinatorial optimization, TSP	survey	survey
The authors describe two basic approaches to using neural networks for optimization. The more popular approach is to formulate a combinatorial optimization task in terms of minimizing a cost function. In this approach, neural network models have been developed or interpreted as minimization machines. The other basic approach is to design competition-based neural networks in which neurons are allowed to compete to become active under certain conditions.					

Cho and Wysk (1993)	selection of dispatching rules for each machine in a FMS	BPN	FMS	flowtime, throughput, tardiness	DR selection
A BPN approach for dispatching rule selection is proposed. The input layer contains system state parameters and the output layer consists of various dispatchin rules. The scheduling function includes three modules (a preprocessor, a neural network, and a multi-pass simulator). The BPN generates the dispatching strategies based on workstation status. The multi-pass simulator then evaluates the best two dispatching strategies in order to select the best strategy to maximize system efficiency. 90 benchmark examples from literature were used as training data. The authors see the robustness of the results as a major advantage of the proposed approach.					
Kovačič (1993)	Markovian NN, timetable	recurrent NN	school timetable	cost: number of unsatisfied constraints	assignment of subjects to rooms and time intervals
In this study, the objective is to schedule a given set of subjects in a limited set of time slots. The proposed recurrent NN is equivalent to inhomogeneous Markov chain and the no. of neurons equals the no. of subjects to be scheduled. Every neuron has (no. of rooms $*$ no. of periods) states which correspond to possible entries in the timetable. The state of the network represents a solution (possibly not acceptable) of the timetable. constraint matrices (one for every additional constraint) are used to model constraint violations. A zero entry means that the respective constraint is satisfied. Entries greater than zero define that a constraint is violated and enter the cost for evaluation function. A Markovian neural network operates by stochastically searching the state space defined by assignments of subjects. Two operators are defined for searching the state space: moving a subject from its current position in the timetable to another position and swapping two subjects. Neurons are asynchronously updated and the probability of transition from one state to another includes simulated annealing method. To increase the speed of simulation the number of successors of the current state in every iteration is reduced. The authors state that although the optimal solution is not always found, the optimal solution is obtained by multiple executions with high probability.					
Lo and Bavarian (1993)	modified 3D-NN	recurrent NN	unrelated parallel machines, multiple TSP	makespan, distance	assignment of operations to machines at a time interval, assignment of cities to positions in a route
The gradient approach of two-dimensional Hopfield network is extended to a three-dimensional matrix, called neural box, in which the third dimension is the time. They use this network to solve the scheduling and multiple traveling salesmen problem. For the scheduling problem, neurons represent a binary 3D-matrix for the assignment of tasks to machines at time t whereas in the TSP problems, neurons encorporate the binary matrix for assigning cities to positions in routes. The energy function consists of two parts, the physical objective function and the constraint functions. Although the simulation results showed that the presented approach yields feasible schedules, too many numbers of neurons and interconnections are required for solving large sized problems.					
Hsu and Yang (1994)	capacitor scheduling: Feeder in Taipei City, knowledge base	SOM	capacitor scheduling, network search problem	total resistive losses + transition cost	schedule cluster
Scheduling knowledge is extracted from training data by using cluster algorithms. Two types of clustering algorithms, hard clustering by Euclidean algorithm and soft clustering by an unsupervised learning neural network (Kohonen's net), are used to classify load curves into groups. In each group, a pre-schedule is obtained by averaging optimal schedules in the group which have been calculated off-line in the first stage. Pre-schedules contain on/off tables of each capacitor at each hour. An uncertainty factor is associated with every on/off decision and by fixing those decisions with lower uncertainty and employing DP to re-schedule those decisions with high uncertainty factors, the final schedule is derived. In the training phase, historical load records are collected and the full DP scheduling is performed off-line. Central concept in this paper is to assist the search task by providing valuable informations about the search space and to give a high quality strategy by which most points in the search space are filtered out.					

Udo and Gupta (1994)	survey	NN	manufacturing processes	survey	survey
The paper presents a state-of-the-art survey of neural network applications in various manufacturing management processes like process control, scheduling, quality control or robotics control.					
Satake et al. (1994)	HNN with Boltzmann machine mechanism, job shop, non-delay schedule	recurrent NN	job shop	makespan	binary variables: assignment of operations to start times
Binary neurons represent assignments of operations to starting times. The proposed model includes Hopfield and Boltzmann machine modes. In the Hopfield mode, the model acts as the Hopfield neural network and generates a feasible non-delay schedule. To seek a better solution, the Boltzmann machine mechanism is activated. In the energy function, only one constraint is included, and the other constraints are reflected in the threshold values. The energy function does not include the objective function or other consraints. Schedules that minimize the energy function are not always feasible, and are not always the best in terms of the makespan, even if feasible.					
Sim et al. (1994)	dynamic scheduling, DR, expert system	BPN	job shop	tardiness, lateness	priority value of jobs
This approach includes a BPN to recognize the individual contributions of traditional DR. The network is incorporated into an expert system which activates the network according to the system state. The expert system will reduce the amount of time required for training by allowing sub-networks to be trained separately. Therefore, 16 sub-networks for the different DR are trained and one is activated by "the expert" depending on the combination of job arrival rate and objective function. The output layer of the sub-network consists of a single neuron whose output is transferred to the output node of the system where it is summed with the output of the sub-networks. Lateral inhibition (similar in concept to Kohonen's map) is applied such that only one of the 16 sub-network output nodes will transfer its output, a priority value for a job, to the system layer. The composite rule can be represented as a simple DT.					
Philipoom et al. (1994)	due date assignment	BPN	HFS	flowtime, tardiness	flowtime
The authors propose a method to set realistic due dates based on the BPN prediction of flowtime of jobs. Input neurons represent job and shop characteristics, and the output is the projected flowtime through the shop, from which the due date assignment can be determined. The training data set is provided my simulation and then used to estimate the parameters in the regression models for the six conventional rules as well as to determine the weights in the neural network case. As scope for future work the authors mention just-in-time systems, were earliness and tardiness increase production cost.					
Kim et al. (1995)	parameter tuning of heuristic,setup times	BPN	single machine	tardiness	parameter of metaheuristic
Based on characteristics of problem instance (no. of jobs, due date tightness and range) the optimal value of the look-ahead parameter k of the heuristic applied to solve the sequencing problem is predicted. 900 test instances were generated and for training and in the simulation tests, 50 random samples for each unique tuple of (no. of jobs, due date tightness and range) are generated, and compared with the results of the heuristics with a constant parameter k. The authors conclude that BPN are a good prediction/preprocessing tool, which may complement a heuristic approach.					
Zhang and Huang (1995)	survey	NN	manufacturing processes	survey	survey
The paper presents a state-of-the-art survey of neural network applications in manufacturing. The objective of this paper is to update information about the applications of neural networks in manufacturing, which will provide some guidelines and references for the research and implementation. In addition, NN approaches are compared to knowledge-based expert systems.					
Ravikumar and Vedi (1995)	reconfigurable parallel processors, Boltzmann machines, directed acyclic task graph, critical path	recurrent NN	reconfigurable parallel processors	total overall execution time	assignment of tasks to processors

The scheduling problem investigated consists of several sub-problems and is solved heuristically: 1. Estimation of no. of processors to maximize processor efficiency via critical-path method or Boltzmann machine approaches. 2. Assignment of tasks to and scheduling them on processors to minimize total completion time via a list scheduling algorithm. 3. Link allocation among processors to minimize total communication time through an iterated heuristic procedure. Neurons represent a binary matrix for assigning tasks to processors. First, the assignment sub-problem is solved by a single Boltzmann machine. Due to long computation times, a parallel algorithm is used by partitioning the task graph into several sub-graphs and solving them concurrently with parallel Boltzmann machines. The cost function to be minimized is a consensus function including total execution time and total communication time. In conclusion, the Boltzmann machine provides better solutions than heuristic in some cases but requires more computation time. To overcome this drawback, decomposition and concurrency is introduced.					
Smith et al. (1996b)	HNN, car sequencing problem (CSP), separation rules for minimum distances between cars	recurrent NN	sequencing cars of different car models along an assembly line	violations of separation rules	assignment of car models to positions in sequence
Several HNN approaches are presented which guarantee feasibility of solutions. Further modifications are proposed to improve the solution quality by permitting escape from local minima in an attempt to locate the global optimum. Neurons represent a binary matrix defining if a car in a position belongs to a model type. The results are compared with those of the steepest descent and simulated annealing approach. The authors conclude that one of the proposed modified neural method matches well the performance of the best of the heuristics (SA), and even outperforms it as the problem size increases. Furthermore, the NN approach is generic and can be applied to several different optimization problems.					
Sabuncuoglu and Gurgun (1996)	modified HNN, SA	recurrent NN	single machine, job shop	makespan, tardiness	permutation matrices for each machine
An additional "external processor" is added to a HNN to heuristically force the network to converge to a feasible final solution in a given number of iterations. Neurons represent the binary permutation matrix that equals the sequence of operations on a specific machine. In the proposed model, the feasibility constraints of the problem are not included in the energy function and both feasibility and cost calculations are carried out by the external processor to simplify the network. The network also possesses a "competition" property to allow the jobs to compete with each other to get the first available position in the sequence. A suggestion for further research is to develop hybrid approaches that combine OR techniques and neural network approaches.					
Smith et al. (1996a)	merge of novel self-organizing NN with a HNN	hybrid approach: SOM + recurrent NN	broad class of 0-1 optimization problems, sequencing problem from car manufacturing industry	miscellaneous	binary permutation matrix
The first stage of the hybrid neural approach is a modification of Kohonen's Self-Organizing Feature Map, operating in multi-dimensional weight space (a unit hypercube), rather than two-dimensional Euclidean space. Variables of the optimization problem, are represented as the network's weights. The SOM attempts to optimize the objective function via competition and inhibition between the nodes of the network. The winning node and its neighbourhood are defined in terms of the objective function of the optimization problem, rather than the spatial distances between nodes. The second stage involves the minimization of an energy function via a HNN in order to restrict the convergence of the network to the feasible solution space. Since the optimality of the network is a consideration of the SOM, the energy function consists only of a single feasibility term so that the usual trade-off problems encountered with such energy functions are avoided. Neurons define binary permutation matrix representing a variety of 0-1 optimization problems. Kohonen's definitions of winning neurons and neighbourhoods have been adapted to enable a new application of the updating rule to optimization problems and the energy function of the HNN is simplified. The authors state that HNN typically yields infeasible solutions and involves tedious parameter tuning to obtain optimal solutions, while methods involving self-organization suffer from a lack of generalizability to non-Euclidean problems. In addition, all of these approaches are local optimization techniques only, and until some mechanisms for reaching global minima are incorporated, it will be difficult to match the solution quafity of a heuristic such as simulated annealing.					

Lee et al. (1997)	GA, induced DT, machine breakdown, bottleneck, knowledge base	hybrid approach: DT + GA	job shop	tardiness	DR selection
C4.5 decision tree is used to find the best overall job release rule for a given system state and a GA generates the best combination of rules for dispatching at individual machines. In order to enable the decision tree to generalize from examples, the system is initialized using various job release/dispatching policies as well as the system parameter values. 1200 jobs are used per simulation what is a large number of jobs for the system being modelled, a semiconductor test operation facility, where less than 50 jobs are processed in a shift. The authors point out that it is extremely difficult to measure the characteristics of system parameters at the individual-machine level, and hence to generate training examples for machine learning corresponding to individual machines. That is the reason why they propose a hybrid approach that combines a decision tree with a genetic algorithm.					
Sabuncuoglu (1998)	classification framework	recurrent NN, competitive NN, BPN	survey	survey	survey
A classification framework is presented for HNN and other optimizing networks, competitive networks, and multi-layer perceptrons (BPN). The author concludes from the existing studies, that HNN and competitive neural networks are most suitable for generating optimal schedules, whereas BPN are very effective in selecting scheduling rules and parametrization of the rules or scheduling policies.					
Min et al. (1998)	dynamic scheduling, priority rule selection	competitive NN, SOM	FMS	multi-objective: e.g. flowtime, tardiness, utilization, WIP	heuristic classes
The output nodes (classes) of the competitive neural network learn to recognize the groups of similar input vectors that include the relative objectives of all evaluation criteria together with the current decision rules and the next decision rules. The NN categorizes the training input vectors according to the similarity base which is defined as the similar amount of the differences of the evaluation criteria between the current production interval and the next production interval. Then the scheduler derives the next decision rule from the matching input vectors of a class through a voting method. Kohonen' s learning rule is used to learn to recognize the groups of similar input vectors from a simulation study. As per the authors, future research should consider more decision rules and decision variables, develop a method which systematically searches the optimal number of output nodes of the neural network and further discard unimportant evaluation criteria in order to reduce the effort to achieve multiple system objectives.					
Kim et al. (1998)	dynamic, inductive learning, Kohonen NN, simulation	hybrid approach: SOM + DT	FMS	multi-objective: e.g. flowtime, tardiness, utilization, WIP	heuristic classes, new dispatching rule
Simulation and competitive neural networks are applied sequentially to extract a set of classified training data, called rough scheduling knowledge, which is used to create a compact set of scheduling rules through an inductive learning decision tree. Finally, the pruned decision tree is converted to a set of production rules (called refined scheduling knowledge) which can be further modified by the user if necessary. The setting of the NN builts upon the previous work of the authors and the approach is refined by using a decision tree to automatically generate new dispatching rules from the classes. Simulation is used for the competitive NN with Kohonens learning rule and the classified simulation data is then used for building the C4.5 decision tree. In addition to previously mentioned scope for future research the authors think that developing a systematic knowledge modification method should be developed so that the knowledge always reflects the dynamic operation conditions of the FMS.					
Jain and Meeran (1998)	modified back-error propagation model	BPN	Job Shop	makespan	sequence of jobs for each machine

A tree encoding method is used to enable the direct input/output mapping with a BPN structure, where the inputs to the first n neurons represent the processing times of jobs on all the machines while the inputs to the next n neurons show the precedence order of machines required to process the jobs. The outputs are adopted in order to represent job sequences explicitly for all machines, each machine has n outputs indicating the sequence of the jobs to be processed on that machine. To avoid convergence towards local minima the modified error propagation model applies the jogging mechanism to the network. Jogging involves a random alteration of the weights to find an alternative path to the desired output. A novel input/output representation permits the number of neurons and interconnections to be minimal. The neural network performs the optimization itself, thus not relying on alternative methods and the use of training data that might include non-optimal solutions. The authors mention that test examples are successful only when they do not vary by more than 20% from any training example. Although this indicates that the generalization capability of neural networks is limited it should be noted that this is one of the first systems that has used a modified BPN model to solve the job shop problem using input/output mappings.					
Smith (1999)	Survey: Neural networks, combinatorial optimization	recurrent NN, SOM	combinatorial optimization	miscellaneous	survey
The focus lies on HNN vs. Kohonen's Self-organizing Feature Map (SOM) but there are also examples presented of Boltzmann, Cauchy, Gaussian machines, counter-propagation network, mean-field theory technique, and winner-takes-all networks. The focus lies on two main approaches: HNN and self-organizing networks. The authors summarize that networks for combinatorial optimization initially dealt with problems of poor solution quality and infeasibility but today they are quite competitive with meta-heuristics such as simulated annealing in terms of solution quality. Future research should consider hybridization of meta-heuristics and NN algorithms as well as more practical optimization problems.					
Arzi and Iaroslavitz (1999)	dynamic, DR selection	BPN, DT	FMS	throughput, tardiness	objective function value
For every decision rule there is a dedicated BPN with an input layer that includes neurons that represent the system state variables, one neuron for each state variable. The output layer has a single neuron, that represents the predicted value of the performance measure at the end of the scheduling period, through which the manufacturing system has been operated by a specific scheduling rule. Simulation is used to generate a realistic testbed representing a FMS operating in a highly random produce-to-order environment. The authors compare the BPN results with the results of a DT and classical dispatching rules and conclude that in a highly random environment the BPN outperformes the DT (which performed as well as, but not significantly better than, the best simple heuristic rule).					
Gupta et al. (2000)	two-tier NN, GA, heuristic selection, 2-stage HFS	hybrid approach: NN + GA	HFS	tardiness	objective function value
Two networks are proposed, namely a one-tier and a two-tier feedforward NN. Neurons at the input layer of the one-tier NN represent characteristic values of the scheduling problem (e.g. no. of jobs, averages and standard deviations of the processing times) and the NN predicts the expected number of tardy jobs and the respective rank for 20 different scheduling heuristics. The output of the one-tier NN is fet to a second NN, as the accuracy in finding the best scheduling rule was not sufficient. Therefore, the two-tier NN maps the relationship between the five best algorithms selected with the one-tier network to the algorithms that were actually the best for a given problem (which does not need to be amongst the top 5 heuristics proposed by one-tier NN). 6000 problems were considered in a simulation study. A GA approach replaces the classical backpropagation learning rule with weights being tranformed into genes of the GA. The fitness evaluation then compares the predicted no. of tardy jobs of the NN aproach with the actual no. of tardy jobs derived by the respective heuristic. As a key advantage the authors point out that the architecture of our proposed neural networks is independent of the size of the problem and has constant input-output lengths. Thus, with a given set of problem characteristics, a neural-network approach can be used to select a scheduling heuristic for largesize problems.					
El-Bouri et al. (2000)	job sequencing, heuristics	BPN	single machine	flowtime, tardiness, cost functions	problem class, priority value of all jobs per class

A two step approach is proposed where a first BPN performs a classification of problem instances and a second BPN then, which is trained specifically for sequencing problems of that particular category, performs the sequencing of jobs. Each job is represented by an input vector which holds information particular to that job and in relation to the other jobs in the problem. The output unit assumes values that are in the range of 0.1 to 0.9, the magnitude being an indication of where the job represented at the input layer should desirably lie in the sequence. To improve the solution quality, a local search (neighbour swap scheme) is performed as the average deviation from optimal position is around 1.5. Optimal solutions are generated by complete enumeration and dynamic programming. The authors compare good, specialized heuristics with the NN approach and mention that it is unlikely to be noticeably better than a good heuristic developed specifically for the purpose and that training and testing require a lot of time. Nevertheless, the advantage of such an approach is that it allows what amounts to a "customized" heuristic to be established for problem subsets and various objectives without having to deduce an algorithm in advance and the development time for a NN approach is much shorter than that needed to evolve a holistic heuristic procedure.					
Lee and Shaw (2000)	real time sequencing, knowledge base, printed circuit board manufacturing	hybrid approach: GA + bi-directional competitive NN	permutation flow shop	makespan	weight matrix representing precedence between pairs of jobs
The proposed approach involves three stages which may be followed sequentially or in parallel: the initial learning stage, the implementation stage, and the knowledge refinement stage. In the initial learning stage the acquired sequencing knowledge (i.e., ideal sequencing pattern) is stored in a long-term memory. In the implementation stage, this knowledge is used to make a real-time sequencing decision. Even though the NN has not seen a set of jobs that arrive at the flow-shop, it can utilize the ideal sequencing pattern stored in the longterm memory to make intelligent sequencing decisions. In the knowledge refinement stage, the NN continuously adapts to the changing manufacturing environments in which a new job type may enter the flow-shop. As a modification of the implementation stage, a hybrid aproach starts with an NN that consists of two fully connected layers (one for preceding jobs and one for succeeding jobs) and the weight matrix associates the desirability of sequences between jobs with the magnitude of weights between jobs, and is constructed before the implementation stage. Afterwards, a heuristic weight adjustment guarantees feasible solutions. After the NN generates a population of initial sequences, a GA procedure is used to improve upon the solutions. Johnson algorithm and NEH + GA, SA and local search solutions are generated as training data. The authors claim that their approach is relatively easy to implement, guarantees feasible sequences, and performs very well in both the solution quality and computational time.					
Yang and Wang (2000)	constraint satisfaction adaptive NN, improvement through heuristics	hybrid approach: recurrent NN + heuristic	generalized Job Shop	makespan	start time, sequence constraint violation, resource constraint violation
A constraint satisfaction adaptive NN is proposed where, for a predefined expected makespan, a feasible schedule is derived by defining start times of jobs. Neurons represent start times of jobs, resource constraints, and resource constraints and are arranged in two layers (the first for starting time blocks, the second for sequence and resource constraint blocks). After an initial feasible schedule is found, improvement heuristics are used to find good solutions. A difference to other constraint satisfaction NN is that neural units' connection weights and biases need not be prescribed in advance before application of the networks to a particular problem. A new heuristic is proposed to adapt the weights asynchronously. The authors emphasis that if only the NN is used, the quality of feasible solutions obtained depends on the choice of an expected makespan. When the expected makespan is suitably chosen, the desired objective can always be achieved. But when the the expected makespan is set too loose the feasible solution may be not a good solution, and when it is set too tight or shorter than the optimum, the feasible solution cannot be obtained. Also, there may appear the phenomenon of nonconvergence among many runs. When applying improvement heuristics, this problem is overcome and good solutions are found even if the expected makespan is equivalent to $+\infty$.					
Park et al. (2000)	setup time, look-ahead parameters of heuristic	hybrid approach: BPN + heuristic	identical parallel machines	tardiness	heuristic parameter

Two BPN are presented that each predict the optimal value of one of the two "look-ahead" parameters of a scheduling heuristic. Training data is generated by simulation: For each combination of parameter 1 and 2, the problem is heuristically solved, then the value of the sum of the weighted tardiness is computed. All parameter values which lead to minimum values of the objective function are identified and the average is taken for both. For each combination of factors, there are 10 averages of parameter 1 and 2. The average of these averages is taken for the optimal value for each problem. The instance set with the optimal values is used for training of NN. Different research possibilities are suggested. One possibility is applying a NN to capture the problem characteristics so that the first NN generates the values of the problem characteristics, and then a second NN uses the results as input values to estimate the parameter values.					
Yang and Wang (2001)	constraint satisfaction adaptive NN and heuristics combined approach	hybrid approach: recurrent NN + heuristic	job shop	makespan	start times, sequence constraint violation, resource constraint violation
The NN model is the same as in the earlier work of the authors. A new heuristic based on the property of non-delay schedules is presented that, together with one of the heuristics proposed by the authors earlier, can be combined with the NN to form a new hybrid approach for job shop scheduling problems. In the hybrid approach, the NN is used to obtain feasible solutions, a weight adaption heuristic is used to accelerate the solving process of constraint satisfaction adaptive NN and guarantee feasible solutions, and the new heuristic is used to obtain the non-delay solution from the feasible solution obtained by NN with determined orders of operations.					
Yu and Liang (2001)	expanded job shop, hybrid GA- constraint NN, constraint NN with gradient search	hybrid approach: GA + recurrent NN	job shop	makespan, tardiness	start times, sequence constraint violation, resource constraint violation
A new constraint NN is presented with three types of neurons representing start time restrictions, depicting precedence and processing restrictions. Neurons are arranged in three blocks for sequence, resource and job constraints check and a hybrid mode of synchronous and asynchronous updates of neurons is utilized. A second constraint NN is proposed with the modification that a gradient search algorithm is applied to optimize start times for a fixed sequence and given energy function. The proposed hybrid approach uses a GA to initialize the NN with permutations for operations sharing the same resources and the NN then derives the fitness function value by optimizing the start times of operations. The authors state that the hybrid approach outperforms other state-of-the-art heuristics for job shop problems.					
Priore et al. (2001b)	DT, case-based reasoning, DR selection, real time scheduling, knowledge base	kNN, DT	FMS	tardiness, flowtime	DR selection, new dispatching rule
The kNN algorithm is utilized to select suitable dispatching rules based on system attributes. Weights of the kNN are adjusted by a GA. As a second scheduling method, a C4.5 decision tree is built to generate new dispatching rules for given system attributes. Training data for both methods stems from simulation studies with 16 dispatching rule combinations. The authors suggest that future research might include consideration of more decision types in the proposed FMS as well as new control attributes that are a combination of those that were initially proposed.					
Chen and Huang (2001)	competitive learning rule for HNN, multiprocessor problem with no process migration, constrained times, limited resources	recurrent NN	job shop	constraint satisfaction, lateness	assignment of jobs to machines in a time interval

The authors propose a competitive HNN by including a competitive deterministic learning mechanism to update the neuron states in the HNN based on a winner-take-all mechanism. Neurons represent 3-dimensional binary decision variables denoting if a job is arranged to execute on machine at a certain time. Additionally, the HNN employs the deterministic rule to update the neuron state change. An energy function designed to illustrate the timing and resource constraints is proposed. HNN can be employed to obtain the weighting and threshold matrices, then the competition process can be applied to obtain the solution. As per the authors, a competitive learning rule can not only reduce the time consumed in obtaining coefficients but also obtains an effective and sound solution. Unsolved problems exist regarding weight initialization of the HNN as well as overcoming local minima.					
Priore et al. (2001a)	discrete Simulation, DR, dynamic scheduling	miscellaneous	survey with focus on DR selection, FMS	survey	survey
The authors list the following research gaps: Comparison of the different machine learning methodologies, determination of the optimum number of training examples, selection of an adequate monitoring period, determination of a mechanism or filter to smooth transitory states, generation of new control attributes using an algorithm that can create attributes that are a combination of the initial ones, incorporation of a simulator, and refinement of the knowledge base.					
Wang et al. (2003)	hybrid BPN for HFS	hybrid BPN	HFS	makespan	assignment of jobs to machines
The authors state that a key feature of the NN design is the integration of problem structure and heuristic information in the network structure and solution. They define neurons as job nodes and machine nodes, as well as two nodes between jobs and machines, that represent signals if a machine is empty or if a job is completed on the previous stage and if a job "won" a machine. A heuristic learning algorithm is proposed to iteratively update the weights and thresholds of neurons. The authors summarize the main alterations to classic NN approaches: First, the hybrid architecture utilizes the structure of optimization problems and reduces the number of neurons and links. Second, the search procedure is deterministic and only searches in a feasible space. Third, the computational load is reduced by applying a heuristic learning strategy including deterministic simulated annealing to avoid local minima.					
Agarwal et al. (2003)	task scheduling, precedence relation, hybrid approach	hybrid BPN	identical parallel machines	makespan	assignment of jobs to machines
The authors modify the approach proposed by Wang et al. (2003) to solve an identical parallel machines problem with precedence constraints and pre-emption possibility. They name the hybrid NN as Augmented Neural Network and emphasis that a critical feature is the one-to-one correspondence between the problem structure and the NN structure, thus, allowing the NN to be augmented by the relevant domain specific knowledge in solving the problem. Six different heuristics are considered as starting points for learning and two heuristic learning strategies are proposed. The authors claim that the presented NN is specifically suited to precedence constrained problems but their generality is yet to be explored. Future research can explore the possibility of this approach for other types of optimization problems.					
Li et al. (2003)	rescheduling, functional virtual population	BPN	FMS	machine utility	DR selection
A BPN is used to select the best scheduling rule for a given system state expressed through three system attributes. Training data is provided by a simulation study. If a change of system parameters occurs, new training instances are heuristically generated and a new so called "Functional Virtual Population" is created to determine new ranges of input parameters to be applied for new training of the NN. This approach provides assistance to learn robust scheduling knowledge for manufacturing systems under rationally changing environments. This study focusses on robust scheduling heuristics to minimize rescheduling requirements in production systems by keeping track of changes in the input data for the scheduling problem.					
Feng et al. (2003)	heuristic correction of BPN solution	hybrid BPN	Job Shop	only feasible solution generated	bit encoded feasible schedule

The hybrid bit encoding method is proposed that represents the processing time and processing order of jobs simultaneously with a single bit. The input sample can be represented as tuples, where some terms states the permutation of machines visited by the jobs and others define the processing times for the jobs on the visited machines. So the output from a trained BPN is able to provide the required information regarding what load should be scheduled for each of the machines in the system. In this study, a factury database is used to generate training data. The output of the NN shows the processing order of a waiting job on machines. As the BPN system is employed, a deviation of the outputs from the expected outputs is not unusual, leading to infeasible solutions. Therefore, a heuristic updating rule is proposed to generate feasible solutions from the BPN output. According to the authors, there are limitations of this study. First, the study is limited to the problem of deterministic time-varying demand pattern over a fixed planning horizon. Second, the study only provides a feasible solution and does not perform optimization.					
Priore et al. (2003)	NN, DT, dynamic, DR selection, control attribute extraction, knowledge base	BPN, DT	FMS	tardiness, flowtime	DR selection, new dispatching rule
The authors built upon their earlier work and include a BPN with 12 output nodes representing dispatching rule combinations. The decision tree C4.5 previously proposed is modified to discover new control attributes (basic arithmetic combinations of attributes used for training). If a new attribute appears in a discovered decision rule it is kept, else discarded. Training data for both methods stems from simulation studies The authors summarize that the decision tree performs better than the BPN and that future research should focus on improving the knowledge base.					
Min and Yih (2003)	Competitive NN, Kohonen learning rule, semiconductor fab scheduling knowledge, DR selection for machines and automated material handling	SOM	FMS	multi-objective: flowtime, slack time, remaining processing time	heuristic classes, DR selection
The authors apply a SOM method proposed by Kim et al. (1998) in a two-stage approach: First, grouping is done based on similar system status and then, grouping based on similar performance measures for "system sub groups" is performed. Following the simulation-based data collection model a SOM classifies all instances obtained from the simulation run and categorizes them through training of the network using the current system status of each instance as an input vector. Afterwards, for each category another SOM model using performance measures of each instance as an input vector is trained and each instance is assigned to a certain class, or system sub group, with similar values of current system status and performance measures. After that, the scheduler has the ability to find a matching instance whose expected performance measures are the most similar to the desired performance measures given by the user. The authors point out that the scheduler can be applied at any point of time, but it only finds the decision rules for the next certain time period, which is predefined in the scheduler. Therefore, it would be advantageous if future work enables the scheduler to be used for any time period (duration) as well as at any point of time.					
Akyol (2004)	BPN, fuzzy parameters, Fuzzy membership functions	BPN	permutation flowshop	makespan	job completion time, makespan
Six heuristics are used to generate training data for one month of real production data. For each heuristic, a BPN is trained to predict the completion times of jobs on each machine. Rather than determining the optimal sequences directly by the network, the BPN indirectly predict the makespan for the next production period for different heuristics so that the most promising heuristic can be chosen. In addition, a method is proposed to determine fuzzy completion times, job waiting and machine idle times. Triangular fuzzy numbers are used to represent fuzzy completion times. Job waiting and machine idle times are considered as triangular or trapezoidal fuzzy numbers. The author states that all selected algorithms are successful in predicting the makespan so that such predictions can be helpful in developing appropriate due dates to enhance customer satisfaction.					
Tang et al. (2005)	dynamic, delta-bar-delta algorithm, steel-making process	BPN	HFS	flowtime, tardiness	priority value of waiting job

Three BPN are used, each for a different objective function. The first six nodes in the input layer correspond to six dispatching rules that have shown good performances in previous research. If one of the six dispatching rule is supposed to be used, then the corresponding input node is set to 0.5, other five nodes are set to -0.5. Node 7 and node 8 correspond to the utilization level and due date demand. The output layer only comprises one node whose value is used to determine the priority of waiting jobs. Simulation is used to generate training data and the delta-bar-delta algorithm is applied during the training, The authors highlight that the proposed method performs well but only considered one type of random event, dynamic job arrivals and future research should include more dynamic changes of the system. Also, they point out that despite the broad practical background, the HFS has not attracted as much attention as other production settings when considering ML approaches.					
Tang and Zhang (2005)	modified HNN, setup times	hybrid recurrent NN	HFS	sum of setup times	assignments of (job) positions to machines at stages
A modification of the HNN is proposed that uses fewer neurons, a different running manner, and includes a local search. Neurons represent binary decision variables for assigning stages to machines and positions. The energy function is simplified and includes one term for the objective and one term for constraint violations. Connecting weights and thresholds are heuristically initialized and an improved sequential asynchronous update mechanism is proposed. After the HNN found a solution, a local search improves the solution quality. The authors state that the hybrid approach outperforms the classical HNN and that the results imply that a more efficient network formulation will exhibit better scaling properties					
Zhao et al. (2005)	constraint neural network, GA	hybrid: recurrent NN + GA	job shop	makespan	start times for given sequence
A hybrid approach is followed where a GA initiates a job sequence and a constraint NN is used to solve constraint violations and to optimize the start times of jobs for the given sequence. The NN is updated using a gradient search algorithm. As per the authors, the biggest advantage of constraint NN is its object-oriented character which makes it easy to implement.					
Shou (2005)	feed-forward NN, priority rule selection per project stage	BPN	project scheduling with renewable resources	makespan	DR selection
The project schedule is generated sequentially and for every stage of the project a BPN is applied that ranks nine different heuristics based on the partial network complexity, partial resource factor, and partial resource strength to select the next activity. An output value less than 0.5 means that the corresponding priority rule is discarded, a value approximating 1.0 that it is recommended. in case of ties, a priority rule is chosen randomly. The author states that the BPN outperforms the single dispatching rules.					
El-Bouri et al. (2005)	local search techniques, initial job ranking by NN	BPN	permutation flow shop	flowtime	job priority
A BPN is trained with optimal sequences of small-size problems to predict job priority values based on a job's total and average processing time and the standard deviation of a job's processing time. The output layer has one node and assigns a priority value to each job, the magnitude of the output indicating, where, in a sequence, the job that is represented at the input layer should be assigned. The trained neural network is next best to a modified NEH heuristic at correctly identifying a job's exact optimal location (zero distance). A problem outlined by the authors is that processing times were generated from a uniform distribution whereas in a more realistic setting, normally distributed processing times may be more suitable. Furthermore, the BPN approach might not be applicable for multiple solutions with same objective function value, as it is often the case when minimizing makespan or mean tardiness.					
Araz (2005)	simulation model, BPN, multicriteria decision aid method, PROMETHEE	BPN	Resource constrained system (machines and workers)	multi-objective (10 criteria)	objective function value

For the multi-criteria dual-resource constraint scheduling problem, simulation is used to collect predefined performance measures corresponding to decision rule set and system state variables. Afterwards, a BPN is trained and tested using the input data provided in the data collection step for each selected performance measures. Then, the NN is used to obtain the performance measures for each shop configuration. A shop configuration considers various decision variables such as dispatching rule sets. Finally, the PROMETHEE approach for multi-criteria decision making is applied to evaluate the alternative schedules with regards to all performance criteria. A multi-criteria DRC scheduler is presented that tries to select appropriate dispatching rules by integrating a simulation model, BPN and PROMETHEE. Future research might consdider different configurations of the BPN.					
Li and Olafsson (2005)	decision tree C4.5, DR generation	DT	single machine	makespan	new dispatching rule
The authors propose an approach to use data mining to discover previously unknown dispatching rules by applying the learning algorithms directly to production data consisting of two main phases: 1. data preparation including aggregation, attribute construction, and attribute selection, and 2. model induction (C4.5 DT) and interpretation. The implicit knowledge of expert schedulers is discovered and can be and existing scheduling practices are generalized into explicit scheduling rules by pairwise comparison of jobs. In addition, structural knowledge may be gained that leads to new rules that improve scheduling performance. A future research topic is to extend the approach to consider comparisons between multiple jobs to learn dispatching rules of higher complexity that could potentially be more effective for scheduling than traditional rules.					
Liu et al. (2005)	dynamic dispatching, SVM, radial basis function	SVM	FMS	throughput	DR selection
The FMS scheduling problem is represented as a multiclass classification problem and the authors adopt the one-against-one method to extend the binary SVM to construct the multiclass scheduler to select a dispatching rule at each decision point for all machines. A training set is obtained by executing a number of simulation runs under a specific performance criterion. The authors summarize that the SVM classifier performs well and also can reduce the variance of the throughput.					
Priore et al. (2006)	knowledge base, inductive learning, BPN, and case-based reasoning	kNN, DT, BPN	FMS	tardiness, flowtime	DR selection, new dispatching rule
Building upon their previous studies, the authors compare the case-based reasoning method combining kNN + GA, a BPN, and inductive learning (decision tree C4.5) for dynamic selection of dispatching rules in a FMS. They conclude that the kNN performs better than BPN and inductive learning and that there is still scope for future research regarding the knowledge base refinement.					
El-Bouri and Shah (2006)	DR selection for each machine in a job shop	BPN	job shop	makespan, flowtime	DR selection
Randomly generated problems are scheduled using optimal permutations of three different dispatching rules on five machines. The optimal solution is defined as the particular combination from the set of available dispatching rules that results in the lowest makespan and is determined by enumerating all possible dispatching rule combinations that the job shop can experience. Input nodes of the BPN represent total processing times per machine, variance of processing times for each machine, and the mean routing order for machine while the no. of output nodes equals the no. of machines by the no. of DR. The authors state that the problem representation employed in this research has the disadvantage that it is necessary to construct and train new networks for job shops having other than five machines (as in the presented example). On the other hand, the trained NN is customized for the job shop characteristics it was trained with, and can therefore identify particular dispatching rule combinations more suited for the target application.					
Shiue and Guh (2006a)	adaptive scheduling, feature selection, generalization	hybrid approach: BPN + GA	FMS	throughput, flowtime, tardiness	DR selection, fitness function GA

The hybrid GA/BPN builts upon the authors previous work but in this study, a BPN is combined with a GA instead of a DT. The GA module evolves an optimal subset of system attributes, and simultaneously determines the optimal ANN topology and learning parameters (according to various performance measures), whereas the ANN module, according to the output of the GA module, generates the training examples that were applied to evaluate the fitness of a given subset of system attributes. The neurons in the output layer are representing nine dispatching rules. Training is provided hrough disctrete event simulation and the results are compared with results of a C4.5 DT. The authors suggest sensitivity analysis for system attributes, parameter tuning for the GA and multi-criteria consideration.					
Shiue and Guh (2006b)	hybrid GA/DT approach, knowledge base	hybrid approach: DT + GA	FMS	throughput, flowtime, tardiness	DR selection, fitness function GA
A hybrid GA/DT approach is proposed to develop a DT-based scheduler. It is used to simultaneously evolve an optimal subset of system attributes and determine learning parameters of the DT from a large set of candidate manufacturing system attributes according to various performance measures derived by simulation. For a given feature subset and learning parameters of a DT decoded by a GA chromosome, a C4.5 DT was applied to evaluate the fitness in the GA process and to generate the scheduling knowledge base. This is done by using the C4.5 program for the specific chromosome and then using the DT to unseen data, the performance value of the DT being the fitness function of the GA. Future research should consider further optimization of the GA parameters, providing more training data for special cases and multi-criteria problems as per the authors.					
Geiger et al. (2006)	DR discovery and parallel learning system, GA with trees representing dispatching rules as genes	hybrid approach: DT + GA	single machine	flowtime, lateness, tardiness	new dispatching rule
The authors propose a hybrid approach using a GA for dispatching rule exploration, where a new encoding method is applied: The general structure used to represent dispatching rule functions is a tree. These tree-data structures are manipulated by the learning strategy of the proposed system. This starts with a candidate set of rules that are either randomly or heuristically generated and these rules are passed to a discrete-event simulation model, where the quality of each rule is assessed using one or more quantitative measures of performance. Extending the learning approach to more interesting and realistic scheduling environments is seen as the major area of future research.					
Akyol and Bayhan (2007)	review NN + scheduling	NN	scheduling	survey	survey
The author gives an overview on NN approaches to scheduling problems, distinguishing between stand alone and hybrid approaches as well as between Hopfield type networks, multilayer perceptrons and competitive type networks.					
Alenezi et al. (2008)	support vector regression model, flowtime prediction, multi-resource, multi-product systems	SVM	job shop	flowtime prediction error	due date
The authors propose a support vector regression (SVR) model to predict flowtimes in a multi-resource, multi-project system to derive suitable due dates. The approach is optimized by testing of different kernels and loss functions and training data stems from disrete event simulation. Training data contains data on the status of the production system at the time the order arrived, which are the inputs, and the actual flowtime of the order, which is the output. The prediction error of the SVR model is compared to that of classic time series models (exponential smoothing and moving average) and to a BPN. Results show that the SVR model has lower flowtime prediction error and is more robust and therefore due-date performance can be improved. Adaptive models for SVR are identified as possible area of future research.					
Geiger and Uzsoy (2008)	DR, batch scheduling, GA with trees representing dispatching rules as genes	hybrid approach: DT + GA	batch processor	flowtime, lateness, tardiness	new dispatching rule

In this work, the authors apply their approach proposed earlier to a batch processor problem. The approach automates the process of searching for the best dispatching rule for a given environment by using a GA to search the space of dispatching rules and evaluating their fitness using a simulation model of the production system under study, thus automating the iterative simulation approach. The authors state that the method could easily be integrated in commercial scheduling systems.					
Shiue (2009a)	essential system attributes, various performance criteria	hybrid approach: SVM + GA	FMS	throughput, flow-time, tardiness	DR selection, fitness function GA
As in the previous work of the author, a hybrid ML + GA approach is proposed for dynamic dispatching rule selection. Here, the z-scores data normalisation and GA-based feature selection mechanisms are used for data transformation tasks, then SVM is applied for the dynamic dispatching rule selection classifier. The GA-based feature selection module evolves near the near-optimal subset of system attributes, and simultaneously determines the near-optimal SVM parameters (according to various performance measures), whereas the SVM classifier module, according to the output of the GA, generates the training examples that were applied to evaluate the fitness of a given subset of system attributes. The SVM classifier module therefore has two primary functions. The first is to evaluate the fitness value generated by the GA module and the second is to generate the SVM-based dispatching rule classifier. Training data is provided through simulation and results are compared with a pure SVM classifier. The author states that the current work only designs one dynamic dispatching rule control mechanism for all machines in the system during a given scheduling interval but that in large complex manufacturing systems, multiple decision rules must be determined simultaneously. Also, he points out that supervised learning methods are very time-consuming when it comes to generating sufficient training data through simulation.					
Yang and Yan (2009)	reinforcement learning, dynamic, DR, knowledge base, agent-based approach	reinforcement learning	FMS	tardiness	DR selection
The authors propose a modified Q-learning algorithm, one of asynchronous dynamic programming algorithms for reinforcement learning. To eliminate the poor effect of a large state space in scheduling problems, B–Q learning algorithm combines the basic sequential algorithmic scheme with the Q-learning algorithm. The B–Q learning algorithm causes clustering in the state yielded by the knowledgeable manufacturing system simulator to improve learning efficiency and generalization capability.					
Shiue et al. (2009)	parameter tuning of ML tools through GA	hybrid approach: DT, BPN, SVM + GA	FMS	throughput, flow-time, tardiness	DR selection, fitness function GA
In this work, the authors combine their previously proposed hybrid approaches for dispatching rule selection. The GA-based feature selection mechanism is used for data transformation tasks, optimal system attribute selection, parameter tuning of the ML-algorithms as well as for the selection of the most suitable ML-algorithm for a certain system state. The ML-algorithms evaluate the fitness function of the GA and the selected algorithm (DT, BPN or SVM) is used for selecting the dispatching rule for the next planning period. The authors see future scope in deriving more ensemble methods for ML algorithms.					
Shiue (2009b)	dynamic scheduling knowledge base for different product mix, decision trees, SOM	hybrid approach: DT + GA, SOM, DT	FMS	throughput, flow-time, tardiness	class of product mix, DR selection
The authors extend their previous work to solve scheduling problems where different product mixes require different scheduling rules. In the two-level approach, first, a two-level self-organizing map (SOM) is used to built class labels for different product mixes/system states. After assigning the label, the entire set of training examples with near-optimal system attributes are learned using the DT algorithm and the knowledge base class label is used to construct the knowledge base class selection mechanism. Finally, the DT algorithm C4.5 is reused to individually learn the various classes in the KB training examples in order to generate the dynamic dispatching rule for each knowledge base class. As in the previous work, discrete event simulation is used for generating training data and feature selection by the hybrid DT + GA-based approach for system attributes is performed. As a summery, five key step are executed: 1. training samples generation mechanism, 2. GA/DT-based feature selection mechanism, 3. building a knowledge base class label by SOM, 4. DT-based knowledge base class selection module, and 5. DT-based dynamic dispatching rule selection module. Future research should consider various job inter-arrival times, multiple decision rules per period, and explore reinforcement learning as per the authors.					

Rouhani et al. (2010)	complete enumeration, Levenberg-Marquardt training algorithm	BPN	permutation flow shop	makespan	priority value of a job
The authors use training data obtained from optimal sequence of problems of flow shop scheduling solved by complete enumeration for a BPN. The trained network predicts a priority value for a job and therefore derives a sequence. Input neurons contain operation times of the job under consideration on each machine, the average operation time of jobs on each machine and standard deviation of operation time of jobs for each machine. As complete enumeration is only a valid method for small problems, alternatives for training data generation have to be explored.					
Mouelhi-Chibani and Pierreval (2010)	simulation optimization, dynamic, 2-stage HFS	hybrid approach: BPM + Simulation + SA	HFS	tardiness	DR selection
The approach proposed by the authors selects the most suited DR in real time each time a resource becomes available in accordance with the current system state attributes. New about this method is, that the NN parameters are determined through simulation optimization. The optimization criterion is the expected performance estimated by simulation. The offline training phase consists of following steps: 1. Define the performance criteria, system state attributes, set of DR that can be relevant on each machine. 2. Build a simulation model of the manufacturing system, build a NN, connect the simulation and the NN. 3. Select a simulation optimization approach, use the NN weights as decision variables for the optimization, use the simulation to measure the objective function. 4. Run simulation optimization, collect the resulting NN configuration. In the example provided, simulated annealing for weights optimization and mean tardiness as objective are used. At the end of the off-line optimization process the best weights are kept and used in the NN connected with the simulation model. As simulation optimization is time-consuming, future research should consider distributed simulation and an automated configuration of the NN topology during the optimization phase with well-chosen initial weights.					
Yang et al. (2010)	constraint satisfaction adaptive NN, heuristics, active schedule, non-delay schedule, complexity	recurrent NN	job shop	makespan	start times, sequence constraint violation, resource constraint violation
The NN model proposed earlier is improved by reducing the complexity of the resource constraint block. As usually only a part of all resource constraints with respect to one machine are violated and hence are relevant for feasibility of the schedule, a dynamic mechanism of constructing aRC-block adaptively during each iteration of NN instead of the original static RC-block is proposed. Furthermore, some new improvement heuristics are presented. The authors conclude that future research might focus on further improvement heuristics for the hybrid approach bur also the NN model itself may be further improved. The knowledge that becomes available during the solving process can be further integrated to improve its performance.					
Chen et al. (2010)	SVM, material handling	SVM	FMS	delay cost, distance	binary decision variable: wait/deliver
In this work, the authors propose a SVM classification method to make a decision on weither a material handling system should wait for a next order or immediately deliver. Training samples are generated offline and use a dynamic programming algorithm to evaluate the class of each sample. The scheduling problem related to a no-wait decision can then be solved by optimization algorithms like Branch and Bound, GA, or SA.					
Noorul Haq et al. (2010)	permuation, initial solution by NN as input for GA, NN	hybrid approach: BPN + GA	permutation flow shop	makespan	job priority

The authors use the NN model proposed by El-Bouri et al. (2005). The network is trained with complete enumeration results for small-size problems with 5, 6, and 7 jobs and the trained network is used to solve larger problems with more jobs but same no. of machines. This sequence is used to initialise the GA for further improvement. The input layer of the proposed neural network has 3 times the no. of machines nodes, containing a job's processing times on each of the machines, the average processing times on each of the machines, the standard deviation of the processing times on each of the machines. This design, in which the size of the input layer depends on the number of machines, means that separate and individually trained networks are needed for flow shops having different numbers of machines. The output layer has only one node, regardless of the number of machines. for every job, a priority value between 0.1 and 0.9 is predicted independently. Afterwards, jobs are sorted in increasing order of priority values. Scope for improvement is expected if other neighbourhood search techniques are included, e.g. SA.					
Priore et al. (2010)	SVM, DR selection, real time scheduling, knowledge base	SVM, DT	FMS	tardiness, flowtime	DR selection, new dispatching rule
For the same scheduling problem investigated in their earlier studies, the authors test SVM using the one-against-one method for dynamic dispatching rule selection based on system attributes. The new approach is compared with the decision tree developed in their previous work. The authors conclude that the SVM-based scheduling system is competitive and suggest that another multi-class SVMs might be studied and that a simulator could be usefully incorporated to decide which rule to apply when two or more theoretically correct dispatching rules are provided. Finally, a knowledge base refinement module could also be added, which would automatically modify the knowledge base when major changes in the manufacturing system occur.					
Olafsson and Li (2010)	DT C4.5, DR generation, GA for optimal instance selection for training DT	hybrid approach: DT + GA	single machine	lateness	new dispatching rule
The authors extend their previous work by a new two-stage approach where the inductive learning phase is preceded by the identification of best practices and the corresponding data instances through a GA. The reason is, that direct data mining of scheduling data can at best mimic existing scheduling practices. The classification is then performed by building a C4.5 tree with subsets and the quality is measured by weighted sum of tree length and tree scheduling objective function. Given this classification problem, a learning algorithm is applied to induce a predictive model based on the training data. This model discriminates between the class values (whether a job should be scheduled ahead of another job or not) based on the independent variables (the system data relevant to the schedule). Future research proposals are testing the approach on a real production system, testing different algorithms for the identification of best training sets, and investigating more complex scheduling problems, e.g. flow shops or job shops.					
Choi et al. (2011)	real-time scheduling, DT, DR selection, re-entrant HFS	DT	HFS	flowtime, tardiness, throughput	DR selection
In this study, a decision tree ID3 is used for periodical updates of dispatching rules, where a planning period is determined by change of system state (e.g. no.of jobs, available machines). The scheduling rule has to determine the allocation of jobs to the machines at each stage as well as the sequence of the jobs assigned to each machine. A real-time controller module exchanges the information with the shop floor, monitors the system states, and dispatches jobs according to the rule released by the scheduler. The Scheduler module determines the point of time when a new dispatching rule is to be selected by the DT based rule selector. The DT based rule selector can be constructed using historical data, knowledge from experts or simulations on the performances of dispatching rules under certain system states. Here, training data is generated through simulation. The authors emphasis that more sophisticated DT might improve the performance and that more case studies including different specific production requirements can be performed.					
Guh et al. (2011)	DR selection (local vs. global), SOM, data preprocessing, real-time scheduling, knowledge base	SOM	FMS	throughput, flowtime, tardiness	class of product mix, DR combination selection

The authors built upon their work from 2009 and propose an intelligent multi-controller that uses SOM-based realtime multiple DR selection mechanism to support a real-time scheduling system. Three main mechanisms are incorporated: 1. simulation-based training example generation mechanism, 2. data pre-processing mechanism and 3. SOM-based real time multiple DR selection mechanism. Under various performance criteria over a long period, the proposed approach yields better system performance than the machine learning-based scheduler using the single DR approach and heuristic individual dispatching rules. The authors summarize that multiple DR selection method outperforms single DR selection methods and should be extended to more complex production systems. Also, reinforcement learning should be explored to react to changes during planning periods.					
Agarwal et al. (2011)	GA (for global search) + NN (for local search)	hybrid approach: hybrid BPN + GA	resource-constrained project scheduling	makespan	modified activity list (as input to GA)
This work uses the NN approach described in Agarwal et al. (2003) and combines it with a GA. The authors state that while the GA approach is very effective for global search, the NN-based approach is basically a non-deterministic local-search technique and that hybridizing these approaches will benefit from the complementary advantages of the two approaches, i.e. GA providing the diversification in search while NN providing intensification. GA and NN iterations are interleaved, feeding their best solutions to each other alternately, requiring switching back and forth between the two techniques. This is not straightforward given that the two approaches work quite differently. While GA is a solution-space based approach, NN is a problem-space based approach. In a solution-space based approach, the solution is perturbed from one iteration to the next, using some mechanism (such as crossover and mutation in GA), whereas in a problem-space approach, the problem parameters (e.g. weight vector in a NN which is modified after each iteration) are perturbed from one iteration to another, while using the same heuristic. The hybrid approach uses the NN to provide a modified activity list for GA optimization. An activity list is a precedence feasible list of all activities of the given project and a schedule can be built for a given activity list using either a serial or parallel schedule generation scheme.					
Shiue et al. (2011)	SOM, Wafer fabrication, accelerating KB generation step, Las Vegas filter and data normalization	SOM	FMS	throughput, flowtime	class of product mix, DR selection
In their previous work, the authors propose an intelligent multi-controller that consists of three main mechanisms: 1. a simulation-based training example generation mechanism, 2. a data preprocessing mechanism including feature selection by Las Vegas Filter/Data normalization, and 3. a self-organizing map (SOM)-based multiple DR selection mechanism. In this work, this approach is applied to a FMS and a wafer fabrication system (FAB) and a two-level SOM approach to cluster all the training examples of the simulation outputs to form the knowledge base of the intelligent multi-controller is used. Then, a SOM-based multiple DR selection mechanism inputs the current system status of the shop floor and sends good (but not optimal) multiple DR selection decision signals of the real-time scheduling system. In the FMS case study, the intelligent multicontroller must assign different decision dispatching rules for each manufacturing cell. In the FAB environment, the intelligent multicontroller must simultaneously assign both the input control and the dispatching rules to conform to the FAB's operating characteristics. Again, authors state that reinforcement learning might have potential for further improvement of the approach.					
Ramanan et al. (2011)	BPN, NN solution as start for GA and Suliman heuristic	hybrid approach: BPN + GA	permutation flow shop	makespan	job priority
The authors follow the same approach as presented in Noorul Haq et al. (2010). Here, the sequence determined by the BPN is taken as initial solution for a GA as well as for a heuristic proposed by Suliman (2000). The authors state that the BPN-GA approach has performed better than all other approaches tested and that the present study reveals that there is considerable scope to use other methods of improving solution such as SA etc.					
Shahzad and Mebarki (2012)	DR detection, tabu search, data mining, DT, simulation optimization	hybrid approach: DT + simulation/TS	job shop	lateness	new dispatching rule

The proposed approach starts with a control module that generates an instance database. The optimization module generates tabu search solutions for a subset of these instances, referred as initial training data set and representing a set of good scheduling decisions. In order to identify and extract the characteristics of these good scheduling decisions, a simulation module is employed which associates a performance measure to each decision taken by the optimization module, extracting the relevant scheduling knowledge, which is then stored in a scheduling database and employed by a learning process to construct a C4.5 DT. Instead of restricting the focus on the DR-space, dominance relations of competing jobs are identified, making use of a set of predefined state attributes. Therefore, the data mining based approach discovers previously unknown priority dispatching rules. Future work might focus on attribute selection, different problem sizes of training data and different loading conditions.					
Shiue et al. (2012)	ensemble of classifiers based on GA wrapper feature selection, DR selection, knowledge base, parameter optimization, majority voting	hybrid approach: DT, BPN, SVM + GA	FMS	throughput, flowtime, tardiness	DR selection, fitness function GA
The authors improve their intelligent multi-controller proposed in 2009 that uses ensemble of GA and different ML tools to support an real-time scheduling system. In the previous work, a hybrid GA approach was used to select the most suitable hybrid GA + ML variant for dispatching rule selection. Here, a combination of the different hybrid GA + ML approaches is proposed. To do so, during the training phase, each individual base classifier (denoted as GA + BPN, GA + DT, and GA + SVM classifiers) determines the near-optimal subset of features and learning parameters. Moreover, each individual base classifier is trained independently using its own replicated training data set via a Bagging method, and all constituent base classifiers are aggregated by a majority voting strategy. The authors state that no research has combined feature subset selection and an ensemble of various classifiers such as BPN, DT learning, and SVM to enhance the performance of the resulting ensemble classifier with regard to the real-time scheduling problem. Future work will focus on the ensemble classification methods and might consider reinforcement learning.					
Kechadi et al. (2013)	cyclic/rolling scheduling, makespan per cycle, recurrent NN	hybrid approach: recurrent NN + heuristic post-processor	job shop	flowtime	start times for given sequence
In this work, a cyclic job shop is considered where a set of tasks are executed repeatedly over a probable infinite time horizon. Although the number of task occurrences is infinite, there is a particular cyclic pattern associated with these occurrences. By generating a special framework (or schedule) comprising a pattern of operating sequences, which will be executed repeatedly, the complexity is reduced by seeking out the optimisation of a single schedule. The optimization approach includes a recurrent NN, where the energy function includes a cost function and a penalty term (determined by Lagrange relaxation technique). The recurrent NN is provided with schedules generated by two simple heuristics and then optimizes the starting times for the given sequence. If the steepest descent approach is stuck in local minima, a perturbation phase is activated (similar to simulated annealing). Heuristic post-processing is required as solutions of NN might not be feasible. Other production systems will be considered in future research and the sensitivity to stochasticity will be investigated.					
Manupati et al. (2013)	SVM-based simulation, rescheduling	SVM	FMS	throughput, tardiness	DR selection
The authors examine the performance of the proposed SVM-based approach under two different operational environments of the FMS which are characterized by the uncertainty of demand. The training data is generated by carrying out a simulation run for different DR. The most relevant DR out of several predefined rules is then selected by a SVM module based on the basis of the current states of the system. At each decision point, the one against one method is utilized for the selection of the best rule to be implemented. The candidate getting the maximum number of votes is implemented for the scheduling period. Different production scenarios and SVK configurations will be tested in the future.					
Priore et al. (2014)	survey: selection of DR by ML algorithms	scheduling	survey with focus on DR selection, FMS	survey	survey

The authors point out that missing benchmarks are the biggest problem in comparing ML scheduling approaches. Overall, they state that SVMs and case-based reasoning are the best ML algorithms, followed by inductive learning algorithms as the second best option, and finally neuronal networks have the worst performance. Nevertheless, they highlight that the great advantage of inductive learning is that the knowledge obtained is intelligible, which could be very useful.					
Tripathy et al. (2015a)	directed search optimization, radial basis function NN, directed NN	hybrid approach: BPN + DSO	multi-processor grid scheduling	makespan	assignment of jobs to machines
The authors introduce three novel approaches for the task scheduling problem using directed search optimization. First, task scheduling is framed as an optimization problem and solved by directed search optimization. Next, directed search optimization is used as a new training algorithm to train a three layer feedforward NN and then a Radial Basis Function NN. The authors summarize their key achievements as: Development of the learning method for NN, development of a method for optimization of Radial Basis Function NN, use of directed search optimization in task scheduling, use of directed search optimization-trained NN in task scheduling, and use of directed search optimization-trained Radial Basis Function NN in task scheduling. They also state that the new approaches outperform a GA.					
Priore et al. (2015)	SVM and k-NN case-based reasoning, C4.5-based control attributes generator, knowledge base	hybrid approach: SVM, DT, kNN	FMS	tardiness, flowtime	DR selection, new dispatching rule
In their latest work, the authors combine the ML algorithms with best performance in their earlier studies to an approach for scheduling using SVMs and case-based reasoning (kNN + GA). A generator module of new control attributes is also incorporated (DT C4.5), and this reduces test error obtained with the machine learning algorithms leading to better performance of the manufacturing system. The authors state that the knowledge-based scheduling system is shown to provide the lowest mean tardiness and mean flow time values.					
Tripathy et al. (2015b)	shuffled frog-leaping algorithm, directed NN, Radial Basis Function NN, hybrid approach	hybrid approach: BPN + shuffled frog-leaping algorithm	multi-processor grid scheduling	makespan	assignment of jobs to machines
In this paper, the authors modify their approach published in the same year by replacing the directed search optimization with a shuffled frog-leaping algorithm. The resulting three novel approaches for the task scheduling problem are: First, solving the scheduling problem by shuffled frog-leaping algorithm. Then, making use of shuffled frog-leaping algorithm-trained NN and Radial Basis function NN for the problem of task scheduling. The authors state that their approach results in fast and good solutions.					
Shahrabi et al. (2017)	reinforcement learning, dynamic, Q-learning, variable neighbourhood search, hyper-heuristic	reinforcement learning	job shop	flowtime	parameters meta-heuristic
In this paper, the authors propose a reinforcement learning with a Q-factor algorithm to enhance performance of the scheduling method for dynamic job shop scheduling problem which considers random job arrivals and machine breakdowns. The scheduling method is based on variable neighbourhood search and at every decision point, parameters of the variable neighbourhood search meta-heuristic are improved. Also, a new approach is introduced to calculate reward values in learning processes based on quality of selected parameters. In the Q-learning algorithm, the goal is to find state-action pair value, which represents the long–term expected reward for each pair of state and action. Here, a state is defined as the shop floor condition influencing the scheduling performance represented by two variables: number of job in shop floor and mean processing time of current operations. Eight actions are also defined, each containing variable neighbourhood search algorithm parameters including number of outer loop iteration, number of inner loop iteration and acceptance threshold.					

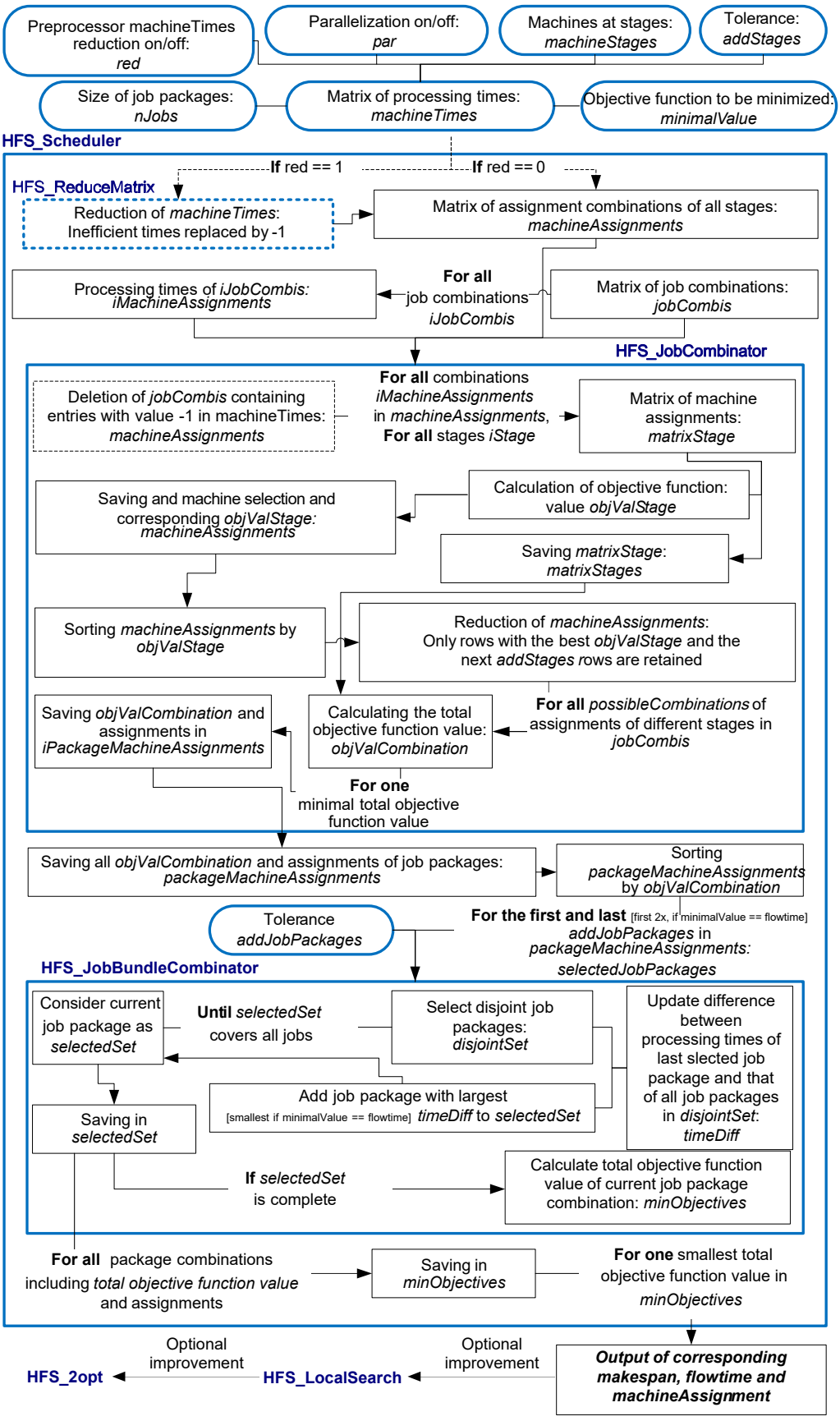


Figure A.1: Flow chart of MAJ heuristic.

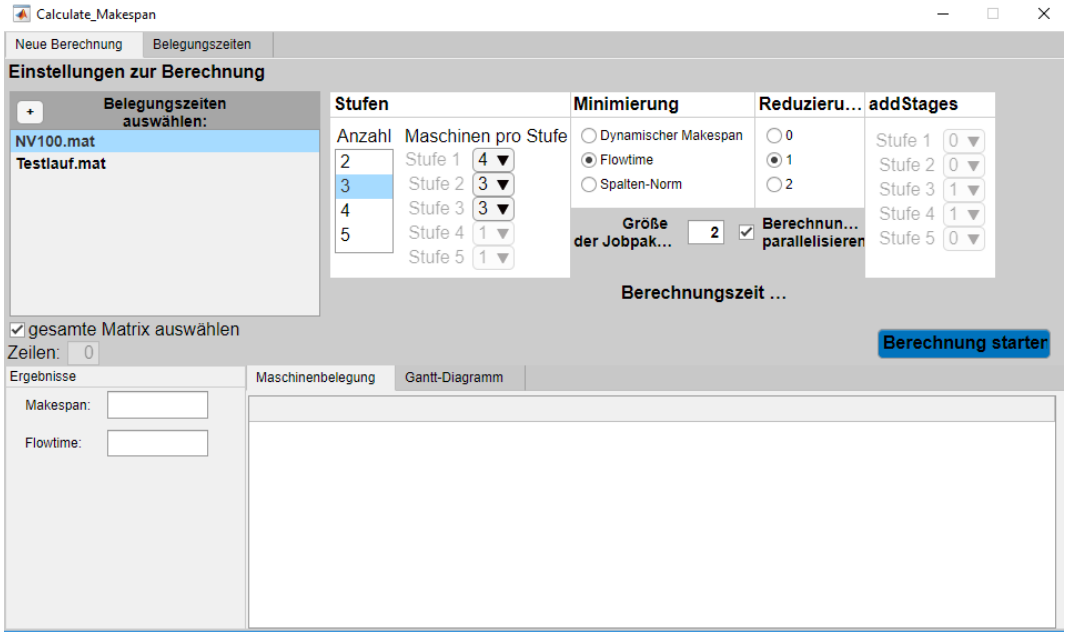


Figure A.2: GUI mask of `MAJ` function call.

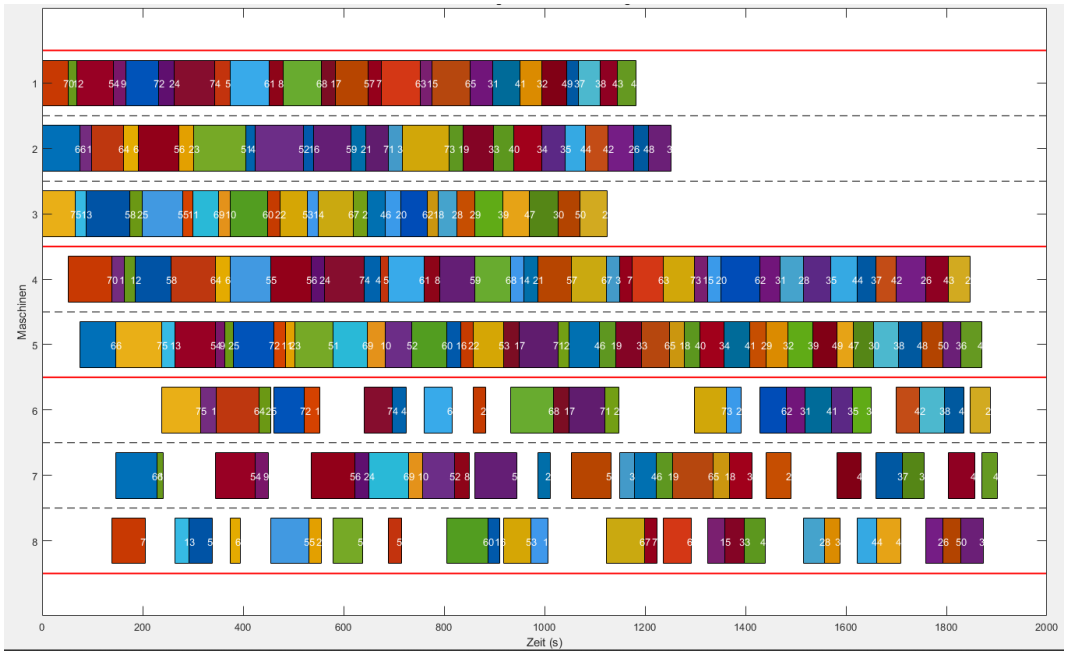


Figure A.3: Example Gantt chart of `MAJ` solution.

References

Agarwal, A., Colak, S., and Erenguc, S. (2011). A neurogenetic approach for the resource-constrained project scheduling problem. *Computers & Operations Research*, 38(1):44–50.

Agarwal, A., Pirkul, H., and Jacob, V. S. (2003). Augmented neural networks for task scheduling. *European Journal of Operational Research*, 151(3):481–502.

Akyol, D. E. (2004). Application of neural networks to heuristic scheduling algorithms. *Computers & Industrial Engineering*, 46(4):679–696.

Akyol, D. E. and Bayhan, G. M. (2007). A review on evolution of production scheduling with neural networks. *Computers & Industrial Engineering*, 53(1):95–122.

Alenezi, A., Moses, S. A., and Trafalis, T. B. (2008). Real-time prediction of order flowtimes using support vector regression. *Computers & Operations Research*, 35(11):3489–3503.

Almeder, C. and Hartl, R. F. (2013). A metaheuristic optimization approach for a real-world stochastic flexible flow shop problem with limited buffer. *International Journal of Production Economics*, 145(1):88–95.

Alpaydin, E. (2009). *Introduction to machine learning*. MIT press.

Araz, O. U. (2005). A simulation based multi-criteria scheduling approach of dual-resource constrained manufacturing systems with neural networks. In *Australasian Joint Conference on Artificial Intelligence*, pages 1047–1052. Springer.

Arizono, I., Yamamoto, A., and Ohta, H. (1992). Scheduling for minimizing total actual flow time by neural networks. *International Journal of Production Research*, 30(3):503–511.

Arzi, Y. and Iaroslavitz, L. (1999). Neural network-based adaptive production control system for a flexible manufacturing cell under a random environment. *IIE Transactions*, 31(3):217–230.

Attar, S., Mohammadi, M., and Tavakkoli-Moghaddam, R. (2013). Hybrid flexible flowshop scheduling problem with unrelated parallel machines and limited waiting times. *The International Journal of Advanced Manufacturing Technology*, 68(5-8):1583–1599.

Behrens, J. T. (1997). Principles and procedures of exploratory data analysis. *Psychological Methods*, 2(2):131.

Bell, J. (2014). *Machine learning: hands-on for developers and technical professionals*. John Wiley & Sons.

Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., and Weglarz, J. (2007). *Handbook on scheduling: from theory to applications*. Springer Science & Business Media.

Boudhar, M. and Meziani, N. (2010). Two-stage hybrid flow shop with recirculation. *International Transactions in Operational Research*, 17(2):239–255.

Brah, S. A. and Loo, L. L. (1999). Heuristics for scheduling in a flow shop with multiple processors. *European Journal of Operational Research*, 113(1):113 – 122.

Bruno, J., Coffman, Jr., E. G., and Sethi, R. (1974). Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17(7):382–387.

Burke, L. I. and Ignizio, J. P. (1992). Neural networks and operations research: an overview. *Computers & Operations Research*, 19(3-4):179–189.

Calvet, L., de Armas, J., Masip, D., and Juan, A. A. (2017). Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs. *Open Mathematics*, 15(1):261–280.

Chapelle, O., Scholkopf, B., and Zien, A. (2009). Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542.

Chen, C., Zhou, B., and Xi, L. (2010). A support vector machine based scheduling approach for a material handling system. In *2010 Sixth International Conference on Natural Computation*, volume 7, pages 3768–3772. IEEE.

Chen, R.-M. and Huang, Y.-M. (2001). Competitive neural network to solve scheduling problems. *Neurocomputing*, 37(1-4):177–196.

Cho, H. and Wysk, R. (1993). A robust adaptive scheduler for an intelligent workstation controller. *International Journal of Production Research*, 31(4):771–789.

Choi, H.-S., Kim, J.-S., and Lee, D.-H. (2011). Real-time scheduling for reentrant hybrid flow shops: A decision tree based mechanism and its application to a tft-lcd line. *Expert systems with Applications*, 38(4):3514–3521.

Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812.

Cui, Z. and Gu, X. (2015). An improved discrete artificial bee colony algorithm to minimize the makespan on hybrid flow shop problems. *Neurocomputing*, 148:248–259.

Dai, M., Tang, D., Giret, A., Salido, M. A., and Li, W. D. (2013). Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. *Robotics and Computer-Integrated Manufacturing*, 29(5):418–429.

de Siqueira, E., Souza, M. J., and de Souza, S. R. (2018). A multi-objective variable neighborhood search algorithm for solving the hybrid flow shop problem. *Electronic Notes in Discrete Mathematics*, 66:87–94.

Defersha, F. M. (2011). A comprehensive mathematical model for hybrid flexible flowshop lot streaming problem. *International Journal of Industrial Engineering Computations*, 2(2):283–294.

Defersha, F. M. and Chen, M. (2012). Mathematical model and parallel genetic algorithm for hybrid flexible flowshop lot streaming problem. *The International Journal of Advanced Manufacturing Technology*, 62(1-4):249–265.

Dios, M., Fernandez-Viagas, V., and Framinan, J. M. (2018). Efficient heuristics for the hybrid flow shop scheduling problem with missing operations. *Computers & Industrial Engineering*, 115:88–99.

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87.

El-Bouri, A., Balakrishnan, S., and Popplewell, N. (2000). Sequencing jobs on a single machine: A neural network approach. *European Journal of Operational Research*, 126(3):474–490.

El-Bouri, A., Balakrishnan, S., and Popplewell, N. (2005). A neural network to enhance local search in the permutation flowshop. *Computers & Industrial Engineering*, 49(1):182–196.

El-Bouri, A. and Shah, P. (2006). A neural network for dispatching rule selection in a job shop. *The International Journal of Advanced Manufacturing Technology*, 31(3-4):342–349.

Elmi, A. and Topaloglu, S. (2013). A scheduling problem in blocking hybrid flow shop robotic cells with multiple robots. *Computers & Operations Research*, 40(10):2543–2555.

Fattahi, P., Hosseini, S. M. H., Jolai, F., and Tavakkoli-Moghaddam, R. (2014). A branch and bound algorithm for hybrid flow shop scheduling problem with setup time and assembly operations. *Applied Mathematical Modelling*, 38(1):119–134.

Fellingham, S. and Stoker, D. J. (1964). An approximation for the exact distribution of the wilcoxon test for symmetry. *Journal of the American Statistical Association*, 59(307):899–905.

Feng, S., Li, L., Cen, L., and Huang, J. (2003). Using mlp networks to design a production scheduling system. *Computers & Operations Research*, 30(6):821–832.

Fernandez-Viagas, V., Molina-Pariente, J. M., and Framinan, J. M. (2018). New efficient constructive heuristics for the hybrid flowshop to minimise makespan: A computational evaluation of heuristics. *Expert Systems with Applications*, 114:345–356.

Figielska, E. (2010). Heuristic algorithms for preemptive scheduling in a two-stage hybrid flowshop with additional renewable resources at each stage. *Computers & Industrial Engineering*, 59(4):509–519.

Figielska, E. (2014). A heuristic for scheduling in a two-stage hybrid flowshop with renewable resources shared among the stages. *European Journal of Operational Research*, 236(2):433–444.

Framinan, J., Leisten, R., and Rajendran, C. (2003). Different initial sequences for the heuristic of nawaz, enscore and ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. *International Journal of Production Research*, 41(1):121–148.

Framinan, J. M., Leisten, R., and Garcia, R. R. (2014). *Manufacturing scheduling systems - An integrated view on Models, Methods and Tools*. Springer.

Garey, M. R., Johnson, D. S., and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129.

Geiger, C. D. and Uzsoy, R. (2008). Learning effective dispatching rules for batch processor scheduling. *International Journal of Production Research*, 46(6):1431–1454.

Geiger, C. D., Uzsoy, R., and Aytuğ, H. (2006). Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. *Journal of Scheduling*, 9(1):7–34.

Gicquel, C., Hege, L., Minoux, M., and Van Canneyt, W. (2012). A discrete time exact solution approach for a complex hybrid flow-shop scheduling problem with limited-wait constraints. *Computers & Operations Research*, 39(3):629–636.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of Discrete Mathematics*, volume 5, pages 287–326. Elsevier.

Guh, R.-S., Shiue, Y.-R., and Tseng, T.-Y. (2011). The study of real time scheduling by an intelligent multi-controller approach. *International Journal of Production Research*, 49(10):2977–2997.

Gupta, J. N. (1988). Two-stage, hybrid flowshop scheduling problem. *Journal of the Operational Research Society*, 39(4):359–364.

Gupta, J. N., Sexton, R. S., and Tunc, E. A. (2000). Selecting scheduling heuristics using neural networks. *INFORMS Journal on Computing*, 12(2):150–162.

Hekmatfar, M., Ghomi, S. F., and Karimi, B. (2011). Two stage reentrant hybrid flow shop with setup times and the criterion of minimizing makespan. *Applied Soft Computing*, 11(8):4530–4539.

Hellstrom, B. J. and Kanal, L. N. (1992). Asymmetric mean-field neural networks for multiprocessor scheduling. *Neural Networks*, 5(4):671–686.

Hidri, L. and Haouari, M. (2011). Bounding strategies for the hybrid flow shop scheduling problem. *Applied Mathematics and Computation*, 217(21):8248–8263.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558.

Hopfield, J. J. and Tank, D. W. (1985). "neural" computation of decisions in optimization problems. *Biological Cybernetics*, 52(3):141–152.

Hopp, W. J. and Spearman, M. L. (2011). *Factory physics*. Waveland Press.

Hsu, Y.-Y. and Yang, C.-C. (1994). A hybrid artificial neural network-dynamic programming approach for feeder capacitor scheduling. *IEEE Transactions on Power Systems*, 9(2):1069–1075.

Hull, T. and Luxemburg, W. (1960). Numerical methods and existence theorems for ordinary differential equations. *Numerische Mathematik*, 2(1):30–41.

Jain, A. S. and Meeran, S. (1998). Job-shop scheduling using neural networks. *International Journal of Production Research*, 36(5):1249–1272.

Jiang, S., Liu, M., Hao, J., and Qian, W. (2015). A bi-layer optimization approach for a hybrid flow shop scheduling problem involving controllable processing times in the steelmaking industry. *Computers & Industrial Engineering*, 87:518 – 531.

Johnston, M. D. and Adorf, H.-M. (1992). Scheduling with neural networks—the case of the hubble space telescope. *Computers & Operations Research*, 19(3-4):209–240.

Jun, S. and Park, J. (2015). A hybrid genetic algorithm for the hybrid flow shop scheduling problem with nighttime work and simultaneous work constraints: A case study from the transformer industry. *Expert Systems with Applications*, 42(15-16):6196–6204.

Kechadi, M.-T., Low, K. S., and Goncalves, G. (2013). Recurrent neural network approach for cyclic job shop scheduling problem. *Journal of Manufacturing Systems*, 32(4):689–699.

Kim, C.-O., Min, H.-S., and Yih, Y. (1998). Integration of inductive learning and neural networks for multi-objective fms scheduling. *International Journal of Production Research*, 36(9):2497–2509.

Kim, S.-Y., Lee, Y.-H., and Agnihotri, D. (1995). A hybrid approach to sequencing jobs using heuristic rules and neural networks. *Production Planning & Control*, 6(5):445–454.

Kim, T. K. (2015). T test as a parametric statistic. *Korean journal of anesthesiology*, 68(6):540.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69.

Kovačič, M. (1993). Timetable construction with markovian neural network. *European Journal of Operational Research*, 69(1):92–96.

Krishnaraj, J., Pugazhendhi, S., Rajendran, C., and Thiagarajan, S. (2012). A modified ant-colony optimisation algorithm to minimise the completion time variance of jobs in flowshops. *International Journal of Production Research*, 50(20):5698–5706.

Kwak, S. G. and Kim, J. H. (2017). Central limit theorem: the cornerstone of modern statistics. *Korean journal of anesthesiology*, 70(2):144.

Lalitha, J. L., Mohan, N., and Pillai, V. M. (2017). Lot streaming in [n-1](1)+ n (m) hybrid flow shop. *Journal of Manufacturing Systems*, 44:12–21.

Lawler, E. L., Lenstra, J. K., Kan, A. H. R., and Shmoys, D. B. (1993). Sequencing and scheduling: Algorithms and complexity. *Handbooks in Operations Research and Management Science*, 4:445–522.

Lee, C.-Y., Piramuthu, S., and Tsai, Y.-K. (1997). Job shop scheduling with a genetic algorithm and machine learning. *International Journal of Production Research*, 35(4):1171–1191.

Lee, C.-Y. and Vairaktarakis, G. L. (1994). Minimizing makespan in hybrid flowshops. *Operations Research Letters*, 16(3):149 – 158.

Lee, I. and Shaw, M. J. (2000). A neural-net approach to real time flow-shop sequencing. *Computers & Industrial Engineering*, 38(1):125–147.

Leisten, R. and Rajendran, C. (2015). Variability of completion time differences in permutation flow shop scheduling. *Computers & Operations Research*, 54:155–167.

Lenstra, J. K., Kan, A. R., and Brucker, P. (1977). Complexity of machine scheduling problems. In *Annals of Discrete Mathematics*, volume 1, pages 343–362. Elsevier.

Li, D.-C., Chen, L.-S., and Lin, Y.-S. (2003). Using functional virtual population as assistance to learn scheduling knowledge in dynamic manufacturing environments. *International Journal of Production Research*, 41(17):4011–4024.

Li, J.-q. and Pan, Q.-k. (2015). Solving the large-scale hybrid flow shop scheduling problem with limited buffers by a hybrid artificial bee colony algorithm. *Information Sciences*, 316:487–502.

Li, J.-q., Pan, Q.-k., and Wang, F.-t. (2014). A hybrid variable neighborhood search for solving the hybrid flow shop scheduling problem. *Applied Soft Computing*, 24:63–77.

Li, X. and Olafsson, S. (2005). Discovering dispatching rules using data mining. *Journal of Scheduling*, 8(6):515–527.

Li, Z.-t., Chen, Q.-x., Mao, N., Wang, X., and Liu, J. (2013). Scheduling rules for two-stage flexible flow shop scheduling problem subject to tail group constraint. *International Journal of Production Economics*, 146(2):667–678.

Li, Z.-t., Liu, J., Chen, Q.-x., Mao, N., and Wang, X. (2015). Approximation algorithms for the three-stage flexible flow shop problem with mid group constraint. *Expert Systems with Applications*, 42(7):3571–3584.

Liang, T.-P., Moskowitz, H., and Yih, Y. (1992). Integrating neural networks and semi-markov processes for automated knowledge acquisition: An application to real-time scheduling. *Decision Sciences*, 23(6):1297–1314.

Liao, C.-J., Tjandradjaja, E., and Chung, T.-P. (2012). An approach using particle swarm optimization and bottleneck heuristic to solve hybrid flow shop scheduling problem. *Applied Soft Computing*, 12(6):1755–1764.

Lin, Y.-K., Huang, D.-H., and Lin, J.-S. (2016). Reliability evaluation of a multistate flexible flow shop with stochastic capacity for multiple types of jobs. *Journal of Manufacturing Systems*, 41:287–298.

Liu, Y.-H., Huang, H.-P., and Lin, Y.-S. (2005). Dynamic scheduling of flexible manufacturing system using support vector machines. In *IEEE International Conference on Automation Science and Engineering, 2005.*, pages 387–392. IEEE.

Lo, Z.-P. and Bavarian, B. (1991). Scheduling with neural networks for flexible manufacturing systems. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 818–823. IEEE.

Lo, Z.-P. and Bavarian, B. (1993). Multiple job scheduling with artificial neural networks. *Computers & Electrical Engineering*, 19(2):87–101.

Looi, C.-K. (1992). Neural network methods in combinatorial optimization. *Computers & Operations Research*, 19(3-4):191–208.

Louridas, P. and Ebert, C. (2016). Machine learning. *IEEE Software*, 33(5):110–115.

Luo, H., Huang, G. Q., Shi, Y., Qu, T., and Zhang, Y. F. (2012). Hybrid flow-shop scheduling with family setup time and inconsistent family formation. *International Journal of Production Research*, 50(6):1457–1475.

Manupati, V., Anand, R., Thakkar, J., Benyoucef, L., Garsia, F. P., and Tiwari, M. (2013). Adaptive production control system for a flexible manufac-

turing cell using support vector machine-based approach. *The International Journal of Advanced Manufacturing Technology*, 67(1-4):969–981.

Marichelvam, M., Prabaharan, T., and Yang, X.-S. (2014). Improved cuckoo search algorithm for hybrid flow shop scheduling problems to minimize makespan. *Applied Soft Computing*, 19:93–101.

Min, H.-S. and Yih, Y. (2003). Selection of dispatching rules on multiple dispatching decision points in real-time scheduling of a semiconductor wafer fabrication system. *International Journal of Production Research*, 41(16):3921–3941.

Min, H.-S., Yih, Y., and Kim, C.-O. (1998). A competitive neural network approach to multi-objective fms scheduling. *International journal of Production Research*, 36(7):1749–1765.

Mouelhi-Chibani, W. and Pierreval, H. (2010). Training a neural network to select dispatching rules in real time. *Computers & Industrial Engineering*, 58(2):249–256.

Naderi, B., Gohari, S., and Yazdani, M. (2014). Hybrid flexible flowshop problems: Models and solution methods. *Applied Mathematical Modelling*, 38(24):5767–5780.

Naderi, B., Ruiz, R., and Zandieh, M. (2010). Algorithms for a realistic variant of flowshop scheduling. *Computers & Operations Research*, 37(2):236–246.

Nakasuka, S. and Yoshida, T. (1992). Dynamic scheduling system utilizing machine learning as a knowledge acquisition tool. *International Journal of Production Research*, 30(2):411–431.

Nawaz, M., Enscore, E. E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91 – 95.

Nikzad, F., Rezaeian, J., Mahdavi, I., and Rastgar, I. (2015). Scheduling of multi-component products in a two-stage flexible flow shop. *Applied Soft Computing*, 32:132–143.

Niu, Q., Zhou, T., Fei, M., and Wang, B. (2012). An efficient quantum immune algorithm to minimize mean flow time for hybrid flow shop problems. *Mathematics and Computers in Simulation*, 84:1–25.

Noorul Haq, A., Ramanan, T. R., Shashikant, K. S., and Sridharan, R. (2010). A hybrid neural network–genetic algorithm approach for permutation flow shop scheduling. *International Journal of Production Research*, 48(14):4217–4231.

Olafsson, S. and Li, X. (2010). Learning effective new single machine dispatching rules from optimal scheduling data. *International Journal of Production Economics*, 128(1):118–126.

Pan, Q.-K., Gao, L., Li, X.-Y., and Gao, K.-Z. (2017a). Effective metaheuristics for scheduling a hybrid flowshop with sequence-dependent setup times. *Applied Mathematics and Computation*, 303:89–112.

Pan, Q.-K., Ruiz, R., and Alfaro-Fernández, P. (2017b). Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows. *Computers & Operations Research*, 80:50–60.

Pan, Q.-K., Wang, L., Li, J.-Q., and Duan, J.-H. (2014). A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation. *Omega*, 45:42–56.

Park, Y., Kim, S., and Lee, Y.-H. (2000). Scheduling jobs on parallel machines applying neural network and heuristic rules. *Computers & Industrial Engineering*, 38(1):189–202.

Philipoom, P. R., Rees, L. P., and Wiegmann, L. (1994). Using neural networks to determine internally-set due-date assignments for shop scheduling. *Decision Sciences*, 25(5-6):825–851.

Pinedo, M. L. (2016). *Scheduling: theory, algorithms, and systems*. Springer.

Priore, P., De La Fuente, D., Gomez, A., and Puente, J. (2001a). A review of machine learning in dynamic scheduling of flexible manufacturing systems. *AI EDAM*, 15(3):251–263.

Priore, P., De La Fuente, D., and Pino, R. (2001b). Learning-based scheduling of flexible manufacturing systems using case-based reasoning. *Applied Artificial Intelligence*, 15(10):949–963.

Priore, P., de la Fuente, D., Pino, R., and Puente, J. (2003). Dynamic scheduling of flexible manufacturing systems using neural networks and inductive learning. *Integrated Manufacturing Systems*, 14(2):160–168.

Priore, P., de la Fuente, D., Puente, J., and Parreño, J. (2006). A comparison of machine-learning algorithms for dynamic scheduling of flexible manufacturing systems. *Engineering Applications of Artificial Intelligence*, 19(3):247–255.

Priore, P., Gómez, A., Pino, R., and Rosillo, R. (2014). Dynamic scheduling of manufacturing systems using machine learning: An updated review. *AI EDAM*, 28(1):83–97.

Priore, P., Parreno, J., Pino, R., Gomez, A., and Puente, J. (2010). Learning-based scheduling of flexible manufacturing systems using support vector machines. *Applied Artificial Intelligence*, 24(3):194–209.

Priore, P., Pino, R., Parreño, J., Puente, J., and Ponte, B. (2015). Real-time scheduling of flexible manufacturing systems using support vector machines and case-based reasoning. *Journal of Economics, Business and Management*, 3.

Rabiee, M., Rad, R. S., Mazinani, M., and Shafaei, R. (2014). An intelligent hybrid meta-heuristic for solving a case of no-wait two-stage flexible flow shop scheduling problem with unrelated parallel machines. *The International Journal of Advanced Manufacturing Technology*, 71(5-8):1229–1245.

Rahmani, D. and Ramezanian, R. (2016). A stable reactive approach in dynamic flexible flow shop scheduling with unexpected disruptions: A case study. *Computers & Industrial Engineering*, 98:360–372.

Ramanan, T. R., Sridharan, R., Shashikant, K. S., and Haq, A. N. (2011). An artificial neural network based heuristic for flow shop scheduling problems. *Journal of Intelligent Manufacturing*, 22(2):279–288.

Rashidi, E., Jahandar, M., and Zandieh, M. (2010). An improved hybrid multi-objective parallel genetic algorithm for hybrid flow shop scheduling with unrelated parallel machines. *The International Journal of Advanced Manufacturing Technology*, 49(9-12):1129–1139.

Ravikumar, C. and Vedi, N. (1995). Heuristic and neural algorithms for mapping tasks to a reconfigurable array. *Microprocessing and Microprogramming*, 41(2):137–151.

Ribas, I., Leisten, R., and Framiñan, J. M. (2010). Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Computers & Operations Research*, 37(8):1439–1454.

Rossi, A., Puppato, A., and Lanzetta, M. (2013). Heuristics for scheduling a two-stage hybrid flow shop with parallel batching machines: application at a hospital sterilisation plant. *International Journal of Production Research*, 51(8):2363–2376.

Rouhani, S., Fathian, M., Jafari, M., and Akhavan, P. (2010). Solving the problem of flow shop scheduling by neural network approach. In *International Conference on Networked Digital Technologies*, pages 172–183. Springer.

Ruiz, R. and Vázquez-Rodríguez, J. A. (2010). The hybrid flow shop scheduling problem. *European Journal of Operational Research*, 205(1):1–18.

Sabuncuoglu, I. (1998). Scheduling with neural networks: a review of the

literature and new research directtions. *Production Planning & Control*, 9(1):2–12.

Sabuncuoglu, I. and Gurgun, B. (1996). A neural network model for scheduling problems. *European Journal of Operational Research*, 93(2):288–299.

Sangsawang, C., Sethanan, K., Fujimoto, T., and Gen, M. (2015). Metaheuristics optimization approaches for two-stage reentrant flexible flow shop with blocking constraint. *Expert Systems with Applications*, 42(5):2395–2410.

Santos, D., Hunsucker, J., and Deal, D. (1995). Global lower bounds for flow shops with multiple processors. *European journal of Operational Research*, 80(1):112–120.

Satake, T., Morikawa, K., and Nakamura, N. (1994). Neural network approach for minimizing the makespan of the general job-shop. *International Journal of Production Economics*, 33(1-3):67–74.

Schmidt, G. (2000). Scheduling with limited machine availability. *European Journal of Operational Research*, 121(1):1–15.

Scholkopf, B. and Smola, A. J. (2001). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.

Scholz-Reiter, B., Rekersbrink, H., and Görges, M. (2010). Dynamic flexible flow shop problems—scheduling heuristics vs. autonomous control. *CIRP Annals*, 59(1):465 – 468.

Schulze, M., Rieck, J., Seifi, C., and Zimmermann, J. (2016). Machine scheduling in underground mining: an application in the potash industry. *OR Spectrum*, 38(2):365–403.

Shahrabi, J., Adibi, M. A., and Mahootchi, M. (2017). A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. *Computers & Industrial Engineering*, 110:75–82.

Shahvari, O. and Logendran, R. (2016). Hybrid flow shop batching and scheduling with a bi-criteria objective. *International Journal of Production Economics*, 179:239–258.

Shahvari, O. and Logendran, R. (2018). A comparison of two stage-based hybrid algorithms for a batch scheduling problem in hybrid flow shop with learning effect. *International Journal of Production Economics*, 195:227–248.

Shahzad, A. and Mebarki, N. (2012). Data mining based job dispatching using hybrid simulation-optimization approach for shop scheduling problem. *Engineering Applications of Artificial Intelligence*, 25(6):1173–1181.

Shaw, M. J., Park, S., and Raman, N. (1992). Intelligent scheduling with machine learning capabilities: the induction of scheduling knowledge. *IIE Transactions*, 24(2):156–168.

Shiue, Y.-R. (2009a). Data-mining-based dynamic dispatching rule selection mechanism for shop floor control systems using a support vector machine approach. *International Journal of Production Research*, 47(13):3669–3690.

Shiue, Y.-R. (2009b). Development of two-level decision tree-based real-time scheduling system under product mix variety environment. *Robotics and Computer-Integrated Manufacturing*, 25(4-5):709–720.

Shiue, Y.-R. and Guh, R.-S. (2006a). Learning-based multi-pass adaptive scheduling for a dynamic manufacturing cell environment. *Robotics and Computer-Integrated Manufacturing*, 22(3):203–216.

Shiue, Y.-R. and Guh, R.-S. (2006b). The optimization of attribute selection in decision tree-based production control systems. *The International Journal of Advanced Manufacturing Technology*, 28(7-8):737–746.

Shiue, Y.-R., Guh, R.-S., and Lee, K.-C. (2011). Study of som-based intelligent multi-controller for real-time scheduling. *Applied Soft Computing*, 11(8):4569–4580.

Shiue, Y.-R., Guh, R.-S., and Lee, K.-C. (2012). Development of machine learning-based real time scheduling systems: using ensemble based on wrapper feature selection approach. *International Journal of Production Research*, 50(20):5887–5905.

Shiue, Y.-R., Guh, R.-S., and Tseng, T.-Y. (2009). Ga-based learning bias selection mechanism for real-time scheduling systems. *Expert Systems with Applications*, 36(9):11451–11460.

Shou, Y. (2005). A neural network based heuristic for resource-constrained project scheduling. In *International Symposium on Neural Networks*, pages 794–799. Springer.

Sim, S., Yeo, K., and Lee, W. (1994). An expert neural network system for dynamic job shop scheduling. *International Journal of Production Research*, 32(8):1759–1773.

Simon, F. Y.-P. et al. (1988a). Integer linear programming neural networks for job-shop scheduling. In *IEEE 1988 International Conference on Neural Networks*, pages 341–348. IEEE.

Simon, F. Y.-P. et al. (1988b). Stochastic neural networks for solving job-shop scheduling. i. problem representation. In *IEEE 1988 International Conference on Neural Networks*, pages 275–282. IEEE.

Simon, F. Y.-P. et al. (1988c). Stochastic neural networks for solving job-shop scheduling. ii. architecture and simulations. In *IEEE 1988 International Conference on Neural Networks*, pages 283–290. IEEE.

Smith, K., Palaniswami, M., and Krishnamoorthy, M. (1996a). A hybrid neural approach to combinatorial optimization. *Computers & Operations Research*, 23(6):597–610.

Smith, K., Palaniswami, M., and Krishnamoorthy, M. (1996b). Traditional heuristic versus hopfield neural network approaches to a car sequencing problem. *European Journal of Operational Research*, 93(2):300–316.

Smith, K. A. (1999). Neural networks for combinatorial optimization: a review of more than a decade of research. *INFORMS Journal on Computing*, 11(1):15–34.

Soltani, S. A. and Karimi, B. (2015). Cyclic hybrid flow shop scheduling problem with limited buffers and machine eligibility constraints. *The International Journal of Advanced Manufacturing Technology*, 76(9-12):1739–1755.

Suliman, S. (2000). A two-phase heuristic approach to the permutation flow-shop scheduling problem. *International Journal of Production Economics*, 64(1-3):143–152.

Tang, L., Liu, W., and Liu, J. (2005). A neural network model and algorithm for the hybrid flow shop scheduling problem in a dynamic environment. *Journal of Intelligent Manufacturing*, 16(3):361–370.

Tang, L. and Zhang, Y. (2005). Heuristic combined artificial neural networks to schedule hybrid flow shop with sequence dependent setup times. In *International Symposium on Neural Networks*, pages 788–793. Springer.

Theodoridis, S. (2015). *Machine learning: a Bayesian and optimization perspective*. Academic Press.

Trabelsi, W., Sauvey, C., and Sauer, N. (2012). Mathematical model and lower bound for hybrid flowshop problem with mixed blocking constraints. *IFAC Proceedings Volumes*, 45(6):1475–1480.

Tripathy, B., Dash, S., and Padhy, S. K. (2015a). Dynamic task scheduling using a directed neural network. *Journal of Parallel and Distributed Computing*, 75:101–106.

Tripathy, B., Dash, S., and Padhy, S. K. (2015b). Multiprocessor scheduling and neural network training methods using shuffled frog-leaping algorithm. *Computers & Industrial Engineering*, 80:154–158.

Udo, G. J. and Gupta, Y. P. (1994). Applications of neural networks in manufacturing management systems. *Production Planning & Control*, 5(3):258–270.

Urlings, T., Ruiz, R., and Serifoglu, F. S. (2010). Genetic algorithms with different representation schemes for complex hybrid flexible flow line problems. *International Journal of Metaheuristics*, 1(1):30–54.

Vaithyanathan, S. and Ignizio, J. P. (1992). A stochastic neural network for resource constrained scheduling. *Computers & Operations Research*, 19(3-4):241–254.

Vignier, A., Billaut, J.-C., and Proust, C. (1999). Les problèmes d'ordonnancement de type flow-shop hybride: état de l'art. *RAIRO-Operations Research*, 33(2):117–183.

Wang, H., Jacob, V., and Rolland, E. (2003). Design of efficient hybrid neural networks for flexible flow shop scheduling. *Expert Systems*, 20(4):208–231.

Wang, S. and Liu, M. (2013a). A genetic algorithm for two-stage no-wait hybrid flow shop scheduling problem. *Computers & Operations Research*, 40(4):1064–1075.

Wang, S. and Liu, M. (2013b). A heuristic method for two-stage hybrid flow shop with dedicated machines. *Computers & Operations Research*, 40(1):438–450.

Watson, J.-P., Barbulescu, L., Whitley, L. D., and Howe, A. E. (2002). Contrasting structured and random permutation flow-shop scheduling problems: search-space topology and algorithm performance. *INFORMS Journal on Computing*, 14(2):98–123.

Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. (2016). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

Yalaoui, F. and Chu, C. (2006). New exact method to solve the pm/rj/σ cj schedule problem. *International Journal of Production Economics*, 100(1):168–179.

Yang, H.-B. and Yan, H.-S. (2009). An adaptive approach to dynamic scheduling in knowledgeable manufacturing cell. *The International Journal of Advanced Manufacturing Technology*, 42(3-4):312.

Yang, S. and Wang, D. (2000). Constraint satisfaction adaptive neural network and heuristics combined approaches for generalized job-shop scheduling. *IEEE Transactions on Neural Networks*, 11(2):474–486.

Yang, S. and Wang, D. (2001). A new adaptive neural network and heuristics hybrid approach for job-shop scheduling. *Computers & Operations Research*, 28(10):955–971.

Yang, S., Wang, D., Chai, T., and Kendall, G. (2010). An improved constraint satisfaction adaptive neural network for job-shop scheduling. *Journal of Scheduling*, 13(1):17–38.

Yao, F. S., Zhao, M., and Zhang, H. (2012). Two-stage hybrid flow shop scheduling with dynamic job arrivals. *Computers & Operations Research*, 39(7):1701–1712.

Yaurima-Basaldua, V. H., Tchernykh, A., Villalobos-Rodríguez, F., and Salomon-Torres, R. (2018). Hybrid flow shop with unrelated machines, setup time, and work in progress buffers for bi-objective optimization of tortilla manufacturing. *Algorithms*, 11(5):68.

Yu, C., Semeraro, Q., and Matta, A. (2018). A genetic algorithm for the hybrid flow shop scheduling with unrelated machines and machine eligibility. *Computers & Operations Research*, 100:211–229.

Yu, H. and Liang, W. (2001). Neural network and genetic algorithm-based hybrid approach to expanded job-shop scheduling. *Computers & Industrial Engineering*, 39(3-4):337–356.

Zabihzadeh, S. S. and Rezaeian, J. (2016). Two meta-heuristic algorithms for flexible flow shop scheduling problem with robotic transportation and release time. *Applied Soft Computing*, 40:319–330.

Zandieh, M., Mozaffari, E., and Gholami, M. (2010). A robust genetic algorithm for scheduling realistic hybrid flexible flow line problems. *Journal of Intelligent Manufacturing*, 21(6):731–743.

Zhang, H.-C. and Huang, S. (1995). Applications of neural networks in manufacturing: a state-of-the-art survey. *International Journal of Production Research*, 33(3):705–728.

Zhao, F., Hong, Y., Yu, D., Chen, X., and Yang, Y. (2005). Integration of artificial neural networks and genetic algorithm for job-shop scheduling problem. In *International Symposium on Neural Networks*, pages 770–775. Springer.

Zhou, D. N., Cherkassky, V., Baldwin, T., and Olson, D. (1991). A neural network approach to job-shop scheduling. *IEEE Transactions on Neural Networks*, 2(1):175–179.

DOI: 10.17185/duepublico/72622
URN: urn:nbn:de:hbz:464-20200902-153418-1