Multi-Agent Reinforcement Learning in a flexible Job Shop Environment: The VCST Case

Tom Beke

Thesis submitted for the degree of
Master of Science: Industrial Engineering & Operations Research

Promotors: prof. dr. El-Houssaine Aghezzaf, dr. Veronique Limère
Mentors: ir. Luiza de Lima Gabriel Zeltzer, Derek Verleye

Faculty of Engineering and Architecture,
Department of Industrial Management
President: prof. dr. El-Houssaine Aghezzaf

Academic year 2012-2013

Multi-Agent Reinforcement Learning in a flexible Job Shop Environment: the VCST Case

Tom Beke

Thesis submitted for the degree of
Master of Science: Industrial Engineering & Operations Research

Promotors: prof. dr. El-Houssaine Aghezzaf, dr. Veronique Limère
Mentors: ir. Luiza de Lima Gabriel Zeltzer, Derek Verleye

Faculty of Engineering and Architecture,
Department of Industrial Management

President: prof. dr. El-Houssaine Aghezzaf

Academic year 2012-2013

Prologue

The completion of a thesis involves many people, without whom the realization would not have been possible.

A first word of gratitude is reserved for Koen Vanharen, production manager at VCST, for the readiness to provide and clarify information about VCST and data for the case, and to free up his schedule to travel all the way to Ghent or to welcome us in Sint-Truiden. This willingness to cooperate was a pleasurable experience and provided a motivational stimulus.

Specifications on the problem description and technical background in which the scheduling problem is embedded, were prerequisite to initiate the project. And of course a project consists of a succession of multiple stages, marked by milestones to evaluate the progress and adjust the course of the project. The meetings with professor Aghezzaf proved to be enriching contributions to the development of the thesis. They were characterized by the low threshold to discuss problems and the personal and empathic contact. I especially want to express my gratefulness for the high degree of autonomy as to maintain a personalized time schedule, especially if at some moments the commitment in student association life was pressing.

The algorithm presented in this thesis was written in C++, which was quite challenging to start coding without any considerable previously acquired programming experience. When I got stuck on the cohesion and the internal logic of this programming language, Luiza was always willing to explain once more and provide. Without the support, I might as well still be coding the algorithm instead of writing these words, and I wish Luiza the best and a lot of happy family moments in the future.

The development of this thesis ran in parallel for a major part with the GUSO board year. Combining the academic, social and musical aspects all together was intensive yet very rewarding. These words are for the full board of GUSO, and in particular Steven, for making it absolutely worthwhile to invest time and energy in making the orchestra a unique meeting place for musicians that are most of all a group of friends. And if on occasions not everything makes sense, than it might as well be senseless in harmony.

Some words I would like to address to my housemates Riet & Steff, for their friendship and support when bad luck stroke, and together with coffee making this world a better place.

To conclude this prologue, the most important insight was brought to me by a dear friend who is in Barcelona now: que sólo se ve bien con el corazón.

Outline

In this thesis, the flexible job shop scheduling problem is addressed to which purpose a reinforcement learning meta-heuristic is proposed, and applied to a realistic middle-sized test case provided by the VCST management. Before the actual dissertation, an abstract was added that summarizes the research highlights and conclusions.

The first chapter offers an introduction to the organization of VCST and sketches the background of the division in Sint-Truiden. Some important concepts are introduced that are used to state the problem description of this thesis. To conclude, the different operations and processes are elucidated, including a description of the machines that are linked to these processes.

The relevant findings from the literature review are presented in chapter two. The evolution of the assembly line is discussed, which leads naturally to the flow shop and job shop design. Robustness and complexity are major issues related to any algorithm, and must be carefully dealt with. The old planning method at VCST is discussed. In the remainder of the second chapter, basic concepts of evolutionary algorithms and reinforcement learning are explained, and the problem specifications are summarized.

In chapter three, the main elements of the algorithm are discussed. Classes, objects and methods are introduced as the building blocks of the algorithm. The two phases of the algorithm are highlighted. In the first phase, an initial schedule is randomly generated. In the second stage, for each agent a set of actions is generated and tested for feasibility, actions are selected, and executed. The objective function is decomposed in several partial functions, and the scope and functionality of the learning mechanisms are explained.

The algorithm is then applied to a set of test instances and the VCST case in chapter 4. The algorithm quality and efficiency are expressed as related to the test cases. The quest for the most suitable objective function is disserted using one selected test instance, and the performance of this objective function is measured against the other test instances. Then the VCST case is addressed and the performance of the meta-heuristic is commented.

The ending chapter 5 presents conclusions in terms of the strengths and weaknesses of the algorithm, and proposes further research opportunities.

Flexible Job Shop Scheduling using Multi-Agent Reinforcement Learning

Tom Beke

Supervisors: prof. dr. E.-H. Aghezzaf, dr. V. Limère, ir. L. de Lima Gabriel Zeltzer, D. Verleye

This paper presents a solution method to the Job Shop Scheduling Problem with work stations in the context of VCST, a producer of gears and shafts for the automobile industry. An improvement algorithm is proposed based on multi-agent reinforcement learning (MARL) in which agents generate 3 types of actions. Learning behavior is stimulated by an ε-greedy strategy and controlled via a global recover point. The algorithm was applied to test instances and a real life test case to measure the performance. An overview of the results and main conclusions are given in this excerpt.

Key words: Flexible Job Shop Scheduling with work stations, Multi-Agent, Reinforcement Learning

I. Introduction

At Volvo Cars Sint-Truiden (VCST), gears and shafts are produced for the automobile industry. The site includes a highly automated Flow Shop and a flexible Job Shop department. Productivity and operational expenses are continuously monitored since they are crucial to maintain a competitive market position, and especially so related to the Job Shop department. Once the customer orders – i.e. a set of jobs with each job consisting of a set of operations - is available for planning, the workload at each work station is known. The objective is to assign every operation of each job to a qualified machine respecting the precedence relations of operations, with zero tardiness and minimum makespan over all the jobs. The data for the real life middle-sized VCST case were provided by the VCST management

Main assumptions are:

- Precedence relations between operations of the same job apply;
- Operations might have a release date;
- Operation processing times are machine-dependent;
- Machines are grouped into functional work stations;
- Single-purpose and single-processor machines that must be qualified for the assigned operations;
- Machine set-up time between operations;
- Operations cannot be interrupted;
- Breakdowns are taken into account using the overall equipment effectiveness.

The remainder of this paper is structured as follows. Section II gives a brief literature review. Section III describes the algorithm. Results are presented in section IV and finally, further research opportunities are suggested in section V.

II. Literature review

The standard reinforcement learning model comprises a set of states, a set of actions, a set of scalar rewards and a transition function [10].

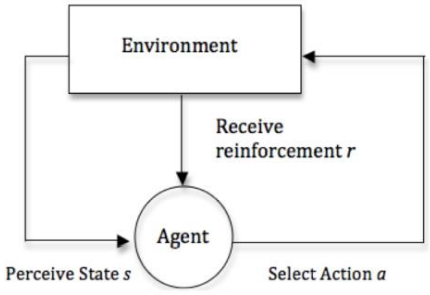

Figure I. Standard reinforcement learning model

At a given instance t, the agent observes a state $s_t \in S$ linked to a set of actions $A(s_t)$. A reward r_{t+1} is assigned after selecting action $a \in A(s_t)$, and the environment is updated to status s_{t+1}. The probability of selecting an action a at instance t is called the agent's policy and denoted as $\pi_t(s,a)$. To learn its behavior, an agent requires feedback.

Multi-agent systems are more complex yet provide several advantages [26]. Computational efficiency is increased by concurrent computation. Enhanced flexibility is provided since the number of agents and their behavior can easily be altered. Uncertainty is reduced when agents can communicate, and the same agent capabilities can be reused in different problems.

A greedy action selection strategy always selects the best action possible. The variance of reward may nonetheless lead to a sub-optimal schedule. A variation is given by the ε-greedy strategy: only in about (100- ε) percent of the cases, the best action is selected. Following a merely greedy strategy, the agent would always select the action with the lowest associated objective value in a minimization problem; now in ε % of the cases a random action will be selected.

III. The MARL algorithm

The structure of the algorithm and composition of the objective function are independent complementary components that are essential to generate a practicable schedule. The algorithm deploys one agent for each of the 12 work stations. For each iteration, every agent generates, evaluates, selects and executes three types of actions, namely Do Nothing (DN), Re-Sequence (RS) and Delegate an Operation (DO). The objective function directs the action selection process. The construction of objective functions suitable for all test instances, and for the VCST case was conducted experimentally starting from eight partial functions.

Key elements are the comprehensive selection and learning mechanisms such as the ε-greedy strategy and setting up a recover point. The recover point moves in each iteration in which a new global minimum objective value is reached and contains the schedule for that configuration. The agents can experiment selecting ε-random actions when the solution gets stuck in a sub-optimal configuration. When a specified number of iterations is reached after the latest recover point was set up and still no improvement is found, the schedule from the recover point is adopted again.

IV. Results

The performance of the algorithm was evaluated based on well-documented test instances for the job shop scheduling problem and a test case provided by VCST. Results revealed that the efficiency in terms of running time and makespan of the schedule is largely

dependent on the number and flexibility of the machines, and the machine-dependency of processing times. The convergence of the solution slows down quickly, as can be read from figure I. After a steep initial decline, the decrease becomes lingering.

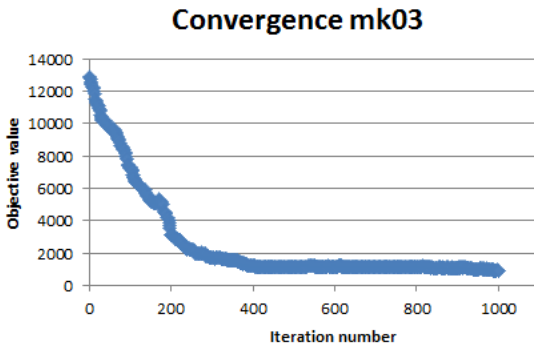

Figure II. Convergence for instance mk03

Finding an appropriate objective function experimentally is problematic, given the variability due to random effects and conflicting behavior between decreasing objective value and decreasing maximum makespan. The best results found with the MARL algorithm are shown in table I for the test instances, and in table II for the VCST case.

Instance	*UB*	*MARL*	*Gap*
Brandimarte mk01	39	41	2.5%
Brandimarte mk02	26	35	22.3%
Brandimarte mk03	204*	263	28.9%
Brandimarte mk07	139	223	55.9%
Hurink rdata - mt06	47*	50	6.3%
Hurink rdata - mt20	1022*	1026	0.39%
Hurink rdata - la01	571	588	2.6%
Hurink rdata - la06	799*	801	0.25%
Hurink rdata - la11	1071*	1073	0.19%
Kacem instance 1	11*	12	9.1%

Table I. Results for test instances

Instance	*UB*	*MARL*	*Gap*
VCST case (OEE = 100%)	*537**	*593*	*10.4%*
VCST case (OEE < 100%)	*(+)*	*757*	*(+)*

Table II. Results for the VCST case

** : upper bound = optimal solution*

+ : data are not available

V. Further research

The strength of the algorithm is the vertical partitioning of the large scheduling problem into subproblems at the work stations with their own schedule. Major issues remain with the unpredictability and variability of the solution, and the construction of an appropriate objective function. To address these issues, the author believes that making the isolated agents communicative and cooperative should be favored over further exploiting the objective function. For doing so, the identification of the critical chain in the schedule and the bottleneck machines are crucial. This will however elevate the level of ingenuity of the algorithm to a more challenging level in the field of artificial intelligence, requiring a major algorithm make-over.

References

[10] A Generic Multi-Agent Reinforcement Learning Approcach for Scheduling Problems, Yailen Martínez Jiménez, 2012

[26] Cawsey, A., The essence of artificial intelligence, Prentice Hall (1998).

Ghent, August 2013

The author **The promotors**

Ir. Tom Beke Prof. dr. E.-H. Aghezzaf dr. ir. V. Limère

List of abbreviations

FFG	Flexible Focused Gearshop
FIFO	First In First Out
FSSP	Flow Shop Scheduling Problem
FFSSP	Flexible Flow Shop Scheduling Problem
JSSP	Job Shop Scheduling Problem
FJSSP	Flexible Job Shop Scheduling Problem
MARL	Multi-Agent Reinforcement Learning
MAS	Multi-Agent System
OEE	Overall Equipment Effectiveness
RL	Reinforcement Learning
STC	Surface Treatment Company
VCST	Volvo Cars Sint-Truiden
WIP	Work In Process

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

1.1 VCST IN THE WORLD & IN BELGIUM

1.1.1 VCST, A GLOBAL SUPPLIER TO THE AUTO-MOTIVE SECTOR

Volvo Cars Sint-Truiden (VCST) is one of the biggest employers in the Belgian province of Limburg since over forty years. As a producer of gears and chassis components for the auto-motive industry, VCST is a global player with a market share of 20% in the market of external gear production via contractors. Today, VCST has industrial sites in the United States, Germany, Mexico, China and Romania while the headquarters are situated in Sint-Truiden, Belgium. VCST employs around 1100 people, slightly more than half of which work in Limburg.

Being a second-tier - as a supplier to the suppliers of the largest automotive manufacturers worldwide - implies a dependency on the global economic climate. Having gone through some precarious years during the economic crisis that struck in 2008 and onwards, VCST has recently set in a revival with orders already filled in up to 80% of the capacity for 2015 and 75% for 2016, thanks to the solid client network based on long-term contracts of 5 till 7 years [1].

In 2009, sales plunged to 92 mio€ with a debt amounting to 200 mio€. In the same year, the Gimv started to shape VCST's future as a dominant stakeholder. Gimv is an independent investment company listed on Euronext Brussels, that specializes in private equity and venture capital, focusing mainly on the European market. Gimv's participation underlines VCST's fundamentally healthy growth perspectives under normalized market circumstances and illustrates the potential to increase its turnover and profitability.

In 2013 the company inaugurated new plants in both China and Romania [2], so to cut back labor costs in low-automated production. Another strategic cornerstone is the focus on innovation and product development. An actual research topic is the noise reduction of operational gears. The headquarters remain firmly anchored on the Belgian site, that is also home to an important research center next to the administration and the highly automated production halls.

1.1.2 THE FJSSP PROBLEM AT VCST: A MARL–HEURISTIC APPROACH

The production hall harbors two distinct areas, each with a distinct organization of the flows. During the week, teams of around 10 operators work 8 hours per day in either the early shift, the late shift or the night shift. If necessary, a weekend shift enables to upgrade the plant's production capacity. The operators are multi-skilled, so they can control various machines.

The **flow shop** consists of several dedicated production lines. A low-volume product type represents an annual production between 60 000 and 80 000 with an operator managing 3 or 4 machines, while a large volume of around 1 000 000 parts per year implies that a team together controls 78 machines. The process quality is carefully monitored using control charts from the statistical process control toolbox: sample values indicating outliers with respect to the warning limits trigger a corrective action from an operator. Since the profitability depends on the volume due to energy and machine costs, and especially the labor cost of the operators, small-volume batches are more expensive than large-volume parts.

The **Flexible Focused Gearshop** (FFG) allows for the production of various product types with smaller lot sizes. More specifically, this department is a ***flexible* job shop**, which is a job shop with similar machines grouped into work centers that we from now on will call work stations. Annual production for product types in the FFG vary between 5 000 and 250 000.

The FFG provides the context of this thesis: a list of jobs, each job associated with a set of operations, needs to be scheduled in the FFG. Each operation of a job must be assigned to one of the machines at a specified work station, with constraints that not every machine at the work station is necessarily qualified for a given operation. Of course, the temporal hierarchy of operations in a job must be respected. This thesis proposes a meta-heuristic approach, based on Multi-Agent Reinforcement Learning.

This scenario is a real-life, medium-sized case based on the information and data provided by VCST. The eventual aim is that the generated schedule can be used as benchmark for the current method of planning at VCST.



1.2 MACHINES & PROCESSES IN THE FLEXIBLE FOCUSED GEARSHOP

The trajectory of a product batch consists of three steps. Parts first undergo a 'soft' treatment at VCST. Then a heat treatment at STC to harden the parts, making them less ductile and for the last phase, after which the parts are shipped back to VCST where the 'hard' operations are executed. With each action from one of the three steps corresponds a work station that consist of a set of similar machines.

1.2.1 SOFT TREATMENT

Work station 1 – Soft turning

The gears obtain the basic shape by scraping off the redundant material.
Machines: MURATEC 20001 and MURATEC 20003 (diameter 300 mm), MURATEC 20002, MURATEC 20004 and MURATEC 20005 (diameter 300 mm).

Work station 2 – Milling

A first notching operation is performed, and the gear circumference acquires the rough relief. The notches are not straight, but squint to improve the grip while making contact.
Machines: PFAUTER 20006, PFAUTER 20007, PFAUTER 20008 and PFAUTER 20009

Work station 3 – Scraping

The purpose of this process is to refine the notching and so to improve the quality of the gears. A better contact reduces the noise during the use of the gears.
Machines: SICMAT 20010, SICMAT 20011 and SICMAT 20012

Work station 4 – Broaching

The central hole in the gear is shaped by drawing a serrated tool through it. This creates the desired pattern on the inside of the gear.
Machines: FORST 20029 and FORST 20030

Work station 5 – Drilling

Extra holes are created in the gears, to connect other components during the assembly in a later stage.
Machines: STAMA 20039 and STAMA 20040, STAMA 20081

1.2.2 HEAT TREATMENT AT STC

VCST works in close cooperation with the neighboring **Surface Treatment Company** (STC), a sister company that is being run independent from the VCST management. Parts undergo first the 'soft' treatments at VCST, then they are shipped to STC, and are then returned to VCST where the 'hard' operations are executed. At STC, the surface of the parts is hardened via a heat treatment. This inevitably alters the shape and notching of the gears, which needs to be corrected for afterwards at VCST. VCST and STC agreed that the time span between a batch reported to be ready for pick-up at VCST, the actual surface treatment at STC and the batch reported to be ready at STC must not exceed 7 days. STC can be linked to a separate (dummy) workstation: **work station 6**.

1.2.3 HARD TREATMENT

Work station 7 – Hard turning
The gears obtain again the basic shape by scraping off the irregularities.
Machines: WEISSER 20037, WEISSER 20098, 6374 MURATEC MS10

Work station 8 – Pressing
In this process, a nib is pressed into the gears.
Machines: SCHMIDT PRESS, VOCO PRESS

Work station 9 – Grinding
Once more, the notching of the outer circumference of the gears is done.
Machines: REISHAUER RZ362, REISHAUER RZ150, VOUMARD

Work station 10 – Inspection 100%
For some batches, the parts need to be inspected by an operator one by one.

Work station 11 – Lasering
On every gear, a code is written: BE for Belgium, the day, the date, and a code referring to the operator that supervises the laser operation at the time of processing.
Machines: TRUMPF 20250, TRUMPF 20257, TRUMPF 20259 and TRUMPF 20260

Work station 12 – Extern FFG
Not every required operation can be performed at VCST, since VCST doesn't dispose of some appropriate (rarely used) machines in Sint-Truiden. These operations are outsourced to an external company. We will group these operations under (dummy) work station 12.

CHAPTER 2. LITERATURE REVIEW AND PROBLEM DESCRIPTION

2.1 THE ASSEMBLY LINE – DEVELOPMENT AND DIFFERENT TYPES

The production hall in Sint-Truiden is divided into two distinct areas, each one with a specific flow organization. In this section, we will briefly describe the evolution of the assembly line, and the need to continuously automate the line and make it better adapted to the product requirements, resulting in a Flow Shop design and the more responsive Job Shop design.

The concept of an assembly line is probably most known through its adoption by Henry Ford, who first applied it to the mass production of the Ford-T model by the beginning of the 20th century. This was a huge improvement in productivity compared to the hand-crafted products. Productivity, and increasingly the flexibility of the assembly line are key factors to the profitability and survival of an enterprise. This is especially so in a global context where new opportunities along with threats arise, which explains why productivity improvements have been a point of interest for many years in the field of Operations Research.

In addition, mass customization allowed customers to purchase the goods conform their requirements. For instance, by choosing from a vast array of options one can highly personalize his new car, hereby considerably increasing the number of different final car versions. To cope with the increased variability, so to keep customers satisfied and maintain or improve the market position, manufacturers need a more flexible production plan.

A lot of research has been done on the **simple assembly line** balancing problem, in which a single model of a given product is being manufactured along the assembly line. Nowadays, to satisfy increasing customer requirements a more flexible schedule is needed. A first improvement is given by the multi-model assembly line [3]: different models are produced in batches on the same line. The logical next step is to reduce the batch size to 1 product: on a **mixed-model assembly line**, every next product can be a different one. For instance, if out of two identical cars one gets leather seats compared to cotton seats for the second one, this will lead to two different final products.

Figure 1. The Single Model Assembly Line [3]

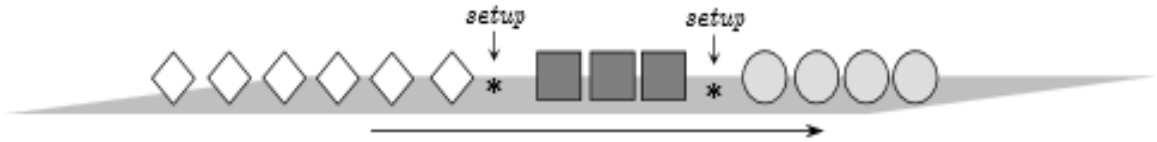

Figure 2. The Multi-Model Assembly Line [3]

Figure 3. The Mixed-Model Assembly Line [3]

2.2 FLOW SHOP VERSUS JOB SHOP

2.2.1 FLOW SHOP DESIGN

In a product-oriented **flow shop design**, every product follows the same trajectory along the assembly line. Machines are dedicated to a specific task and are embedded in the line. Each product that moves over the line visits every single machine in the same machine order. A flow shop design is appropriate for large-volume product families. Similar products are grouped into the same product family and they are produced on the same line, that can easily be automated and run for long periods without human intervention.

A **flow shop scheduling problem** (FSSP) consists of determining the distribution of product processing tasks over the machines on the assembly line and determining the optimal sequence for launching the products on the line. The FSSP can be solved in an exact mathematical way only for the 2-machine variant. An extension to the classical flow shop problem is given by the Flexible Flow Shop Scheduling Problem (FFSSP), where product operations are performed in the same order of machine clusters or work stations. Instead of being assigned to a single machine, an operation is now assigned to one of the machines in the workstation.

2.2.2 JOB SHOP DESIGN

A Job Shop is a generalization of the Flow Shop. In a **Job Shop design**, every product no longer has to follow the same fixed trajectory along the assembly line. Each product follows its own path through the hall and its own specific machine order. Machines are more general-purpose, and are divided in functional groups. The process-oriented Job Shop design allows more flexibility and is suitable when the product portfolio is characterized by high-variety low-volume products. For a job shop the in-process inventory (WIP) is higher than with the FSSP where pull systems can more easily limit the WIP. More change-overs require the operators to intervene more often in a Job Shop design, compared to the Flow Shop design.

The efficient management of a job shop is challenging, as there are multiple interfering product flows in the department. We will from here on associate a job with each product, each job consisting of several operations that are performed on various machines.
If not carefully planned, the multitude of crossing product flows may cause a machine to be idle for long periods waiting for products to arrive, while at some point several jobs might arrive at the same time leading to a long waiting queue in front of the machine, that suddenly is turned into a bottleneck. To avoid this from happening, the sequencing order of jobs on the machines is studied in the **Job Shop Scheduling Problem (JSSP).** The sequencing is made more complex by the precedence relations: an operation of a job can only be processed on a machine if all preceding operations are finished. Hence the schedule must take into account time and resource constraints, and the schedule's performance depends on how well it deals with the Critical Chain as described first by Eliyahu Goldratt [4].

Similar to the relationship between FSSP and FFSSP, an extension to the classical Job Shop problem is given by the **Flexible Job Shop Scheduling Problem** (FJSSP), where jobs pass through work stations. Each job still has its specific order of operations, yet each operation is now linked to a cluster of machines: a work station. An operation must be assigned to one of the machines in the workstation. As a generalization of the job shop scheduling problem, the flexible job shop scheduling problem is equally known to be NP-hard.
The FJSSP problem frequently occurs in practice since often enough several machines are found qualified to perform an operation. The scheduling problem in the FFG at VCST is classified as a FJSSP.

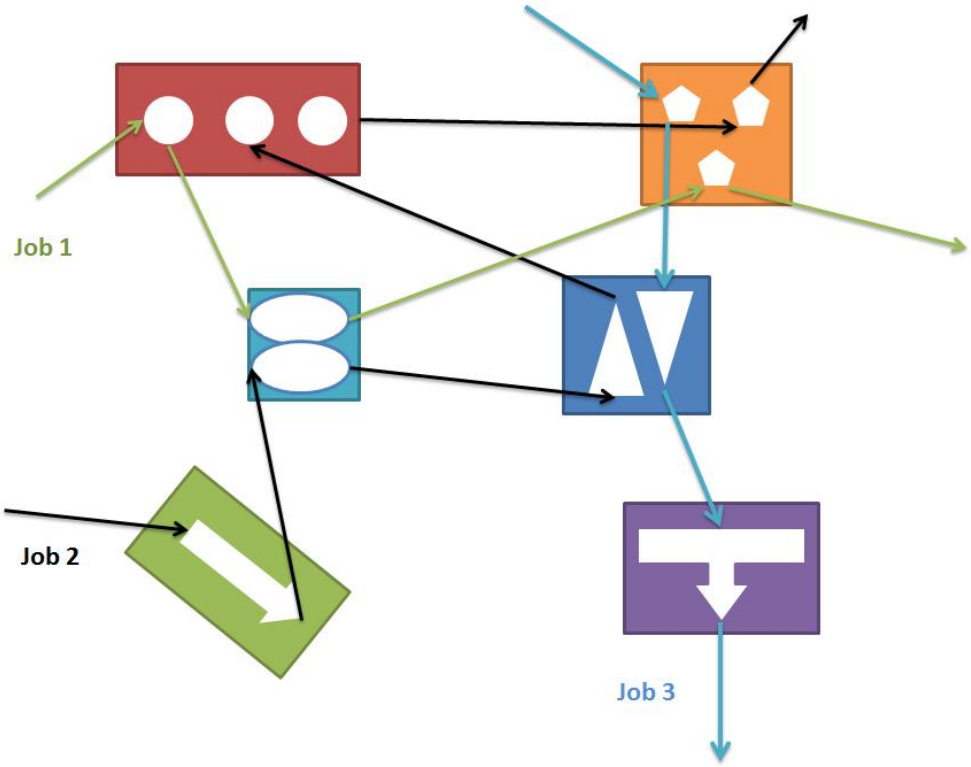

Figure 4. A flexible job shop floor design

2.3 ROBUSTNESS

An optimal solution of the scheduling problem might become useless if a small uncertainty in the data slightly changes the problem. Robustness is a measure for the quality of a solution. In a real-world environment, not all information (stochastic process times, demand forecast, …) is known at the time of composing the schedule. Additionally, a non-preemptive outage can occur, priority can be given to a new-arrived order, existing orders can be cancelled, or the time frame of an order can be modified.

One can read with Policella [5] that " a sub-optimal schedule that contains some built-in flexibility for dealing with unforeseen events, might provide useful characteristics for the execution. Hence, the efficiency of the scheduling techniques employed will depend on the degree of uncertainty of the scheduling information provided at the beginning."

In literature, several classifications of robustness are used. Billaut [6] focuses on data uncertainties: an algorithm is robust whether it is sensitive or not to missing data or wrong data. Another focus lies with the occurrence of unforeseen stochastic events. Robustness can be measured by the deviation from the original schedule (e.g. temporal deviation), the cost incurred, the stability relative to a performance criterion or the number of changes required to get back on track.

Uncertainty can be dealt with in two ways. **Reactive scheduling** is initiated after a disruption of the schedule occurred. In simple rescheduling the processing of operations is delayed if necessary, while keeping the same processing order. More advanced strategies involve the re-sequencing of the unfinished tasks.

In predictive or **proactive scheduling**, techniques are employed to produce an off-line schedule robust to execution time events [7]. Therefore redundancy is implemented in the solution using slack-based techniques. Next to the probability distributions for these events to occur, criticality is a valuable source of information. An operation or resource is critical to the degree of impact on the process. In short, slack time is added to critical parts of the schedule as well as to parts that are likely to require extra execution time. A description of several slack-based techniques can be found in [10].

2.4 COMPLEXITY

For some decades, finding an efficient scheduling algorithm for the Flow Shop Scheduling Problem (FSSP) has been the focus of research. The absence of an optimal exact algorithm has its origins in the computational complexity of the Flow Shop problem. This also holds for the Job Shop scheduling problem, which is in fact a generalization of the FSSP.

For the case in which only 2 machines are considered, an exact algorithm exists and was provided by S.M. Johnson. [8] This problem can be solved in time proportional to n. log(n) for n jobs consisting of multiple operations to be scheduled. In an m-machine flow shop with m ≥ 3**,** one can prove that the problem of finding a shortest-length schedule is NP-complete. NP-completeness is characterized by the absence of an efficient way to find an optimal solution, and by the rapidly increasing computational time with growing problem size so that moderately sized problems can require several million years to be solved exactly [9].

It is widely believed that all problems in the NP-complete class are computationally intractable for two reasons: no efficient algorithm is known for any problem in this class; furthermore, if any single problem in this class could be solved then every other problem would be solvable as well, since NP-complete problems can be linked one to another through a transformation.

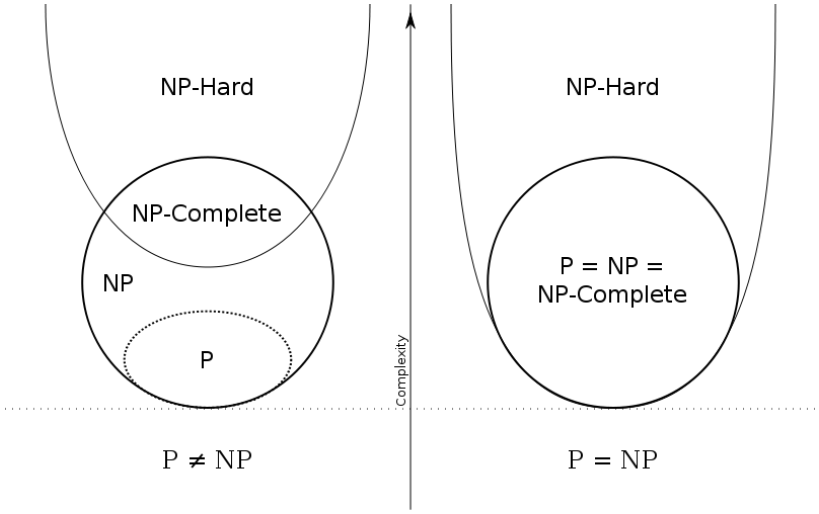

Figure 5: Euler diagram for P, NP, NP-complete, and NP-hard set of problems [9]

Generalizations of the 2-machine Flow Shop problem can turn the problem into a NP-complete one, e.g. with parallel machines, where one type of machine is duplicated. The Job Shop problem proves to be NP-complete even for the 2-machine variant. Hence, an efficient heuristic approach is a preliminary requisite to deal with NP-complete problems.

2.5 THE CURRENT SCHEDULING METHOD AT VCST

The competition is fierce in a globalized world, and striving for cost reduction and optimization through the use of an efficient job schedule is crucial. Finding an efficient schedule is more easily done using software rather than constructing a schedule manually, based on experience yet sensitive to sub-optimal decisions. We already mentioned that the FJSSP is NP-hard, which urges the need for a meta-heuristic approach.

Any newly proposed algorithm must be able to provide a solution for the scheduling problem in a short time and the proposed solution must come as close as possible to the optimal solution, if it has to outperform the old practice. "Short" in terms of computational time is related to the refresh rate of the schedule, for instance new jobs might arrive at VCST or machines can break.

The quality of the solution can be evaluated by comparing the new schedule to the old schedule that was used at VCST. If the new schedule manages to achieve a lower makespan objective value (i.e. complete the jobs in less time), then the algorithm yields an improvement. Another benchmark could be the comparison of the meta-heuristic solution with an exact solution.

Multiple factors make the scheduling problem complex: the number and priority of the jobs, the number of machines, required number of operators, machine break downs, organizational changes to the production hall, changes to the job list and more probabilistic effects that cannot be eliminated. Until now, the scheduling of jobs in the FFG relies on the experience and expertise of the planner, who makes use of macros in Excel. Even if Excel allows to process lots of data in an organized way, in the end there is no guarantee that the solution is actually a good solution – except for the expert judgment of the experienced people at VCST.

The current way of scheduling goes as follows: a client passes on a demand, and the MRP system translates the demand to production commands based on the lot size and the stocks of raw materials. Calculating backwards from the due date, with the estimation of the processing times, the release date is determined. If the priority of a batch is high, it is moved up the queue and processed earlier; otherwise the priority rule that applies is the First In First Out principle (FIFO). Only the launch of product batches at the first work station is scheduled deliberately, afterwards scheduling is based on the priority rules. [12]

Not only is the nature of the scheduling problem complex, the human element also is a cause of inefficiencies. As professional a human can be, he remains fallible. Another problem occurs when the planner gets sick, retires or leaves on holidays and the planner needs to be replaced by a less experienced colleague. To guarantee some degree of continuity, a scheduling algorithm can prove useful as a tool.

2.6 HEURISTIC METHODS TO SOLVE THE FLOW SHOP PROBLEM

> *"A heuristic technique (or simply heuristic) is a method which seeks good (i.e. near-optimal) solutions at a reasonable computation cost without being able to guarantee optimality, and possibly not feasibility. Unfortunately, it may not even be possible to state how close to optimality a particular heuristic solution is."* [13].

Basically, there are three main scheduling concepts: exact mathematically grounded algorithms, heuristic approaches and algorithms supported by machine learning.

Two broad **heuristic categories** are distinguished, namely constructive methods and improvement methods.
A lot of different **constructive heuristics** were developed throughout the years for the Flow Shop problem, more than for the Job Shop problem. This is not surprising, since product customization and flexibility is a relatively young issue.
Johnson's three-stage production schedule is one of the earliest constructive heuristics [14]. Other examples are given by Palmer [15], Gupta [16], Campbell-Dudek-Smith (CDS) [17] and Dannenbring's Rapid Access (RA) procedure [18]. The most effective constructive heuristic for the flow shop problem however is agreed to be the Nawaz-Enscore-Ham (NEH) [19]. Dannenbring came up with two different **improvement heuristics:** RACS (Rapid Access with Close Order Search) and RAES (Rapid Access with Extensive Search). A more recent algorithm was proposed by Suliman [20].

We can also place metaheuristics in the class of improvement heuristics. In this class we find Simulated Annealing, Tabu Search, Genetic Algorithms, Reinforcement Learning algorithms, or any hybrid cross-version of the methods mentioned.
We will now take a better look at the Genetic Algorithms, and Reinforcement Learning. With increasing computational power of computers, the potential for these methods is high. For a more detailed description of the other techniques, we refer to the specific literature for Simulated Annealing [21], Neighborhood Search [22], Tabu Search [23, 24], and Artificial Intelligence [25,26].

2.6.1 GENETIC ALGORITHMS

In a GA, the natural process of genetic evolution is mimicked. An initial set of individuals forms the population. Every individual or chromosome is encoded into a structure that represents its properties. Each individual in this population has a corresponding fitness value: the higher the value, the closer the individual comes to optimality [27], [28].

The initial population undergoes a series of operations and the individuals in it evolve. A complete iteration is called a generation and can be briefly described as follows: a selection mechanism picks individuals of the current population in such a way that an individual's chance of being selected increases with the fitness value. The selected individuals generate new individuals, called the offspring. After this process of crossover, some of the offspring might undergo a mutation. Then the process is repeated, until a stopping criterion is met.

The effectiveness of GAs greatly depends on the correct choice of the encoding, selection, crossover and mutation operators, as well as the probabilities by which they are applied.

2.6.1.1 SELECTION AND REPLACEMENT

The process of replacing old members of the population by offspring is termed a generational scheme. In an elitist generational GA the offspring takes the place of the parents, whereas in a **steady state** GA the offspring replaces the worst performing individuals – i.e. individuals that have the lowest fitness value, under the constraint that the offspring outperforms these worst individuals of course. **Premature convergence** can occur, in case all population members are in essence similar. This problem is overcome by imposing the additional constraint that the sequence of offspring replacing an old individual mustn't be repeated throughout the population.

A popular strategy to select individuals for mating is done using the **tournament**: the worst one out of a couple of individuals randomly taken from the population is added at the top of an empty list, the individual with the highest fitness value returns to the population. The process is repeated until all individuals joined the list. Then, in order of appearance, chromosomes are eligible to undergo actions by genetic operators. **Ranking** of the population according to its fitness value offers an alternative: this basically is a tournament in which the entire population is participating at the same time. [29]

2.6.1.2 CROSS-OVER

Several operators can be used (OX order crossover, UOB or uniform order based, OP or one-point order cross-over, GPX or generalized position cross-over, TP or two-point order cross-over). Two major problems are faced: the offspring frequently has a makespan superior to their parents', and building blocks of winning gene combinations tend to be disrupted by cross-over. Ruben et al. propose four new operators [27], of which the Similar Block Order Crossover (SBOX) seems most promising: common jobs in both parents are only copied in blocks of at least two jobs, then a cutting point is selected and the missing jobs are assigned in order of appearance with the parents.

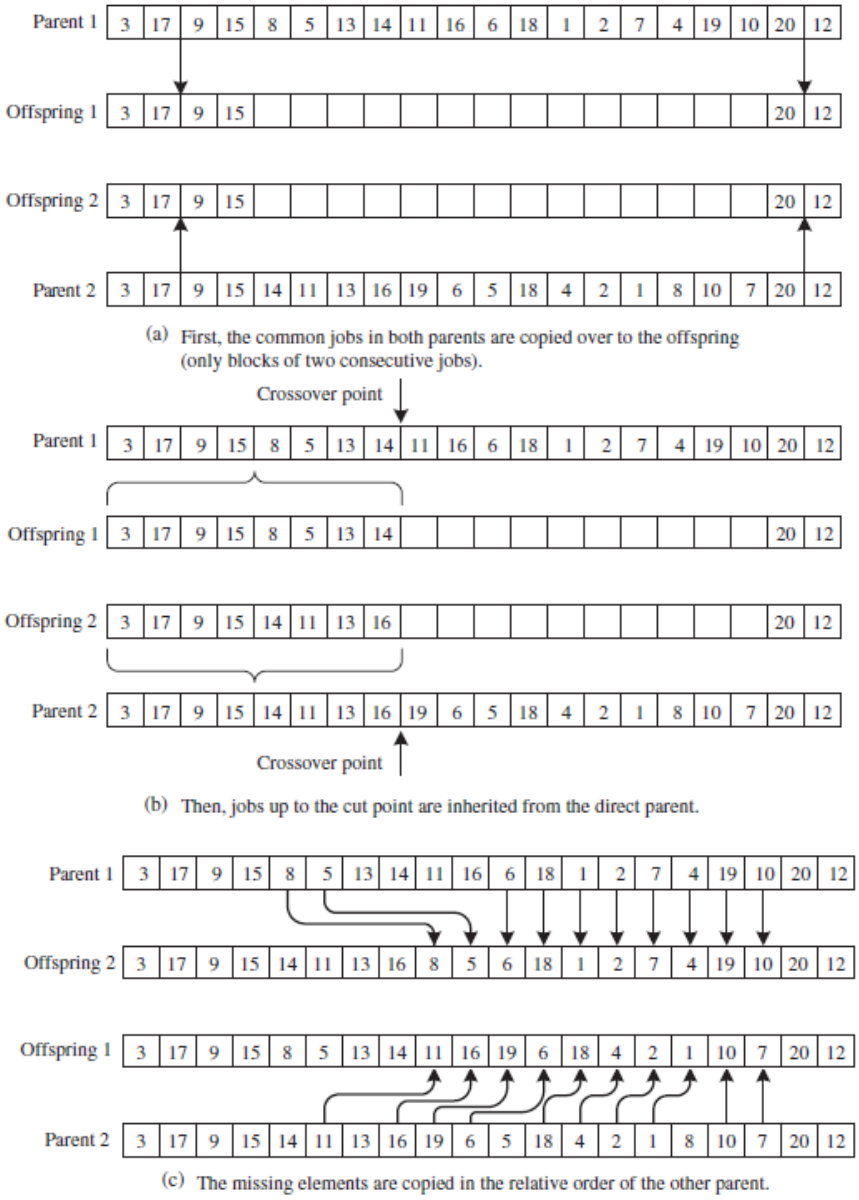

Figure 6. Crossover in Genetic Algorithms [27]

2.6.1.3 MUTATION

Local optimum convergence and the loss of genetic material along with variability can at the same time be tackled by introducing a mutation operator. In the literature, three different operators occupy the scene: *swap mutation* – two positions are selected at random and the corresponding jobs are switched; *position mutation* is a special case of SWAP mutation where the two positions have to be adjacent; and finally *shift mutation* where a job at a random position is inserted at another random position, while for the remainder the job sequence is left unchanged.
The shift mutation has been found to outperform the other operators ([30] , [31]).

2.6.1.4 RESTART SCHEME

Even when mutation is enabled, the algorithm can converge to a local optimum when the populations achieved a sufficiently low diversity. To escape this sub-optimum, a restart procedure can be designed, for example by withholding only part of the sub-optimal population unaltered, while the remaining part undergoes a series of mutations hence increasing diversity before we the algorithm continues.

2.6.1.5 STOPPING CRITERIA

A popular criterion applied in GAs is two-fold: a trade-off between convergence and running time is made. On one hand, it makes no sense to continue executing the algorithm if the best solution found does not improve significantly anymore (i.e. the improvement is less than a certain percentage) after a pre-determined minimum number of generations created: a state of convergence is reached; on the other hand, time is costly so the algorithm is disrupted after a maximum number of iterations. The upper and lower bound on the number of iterations and the threshold for the percentage of improvement can be derived from a convergence study.

2.6.2 MULTI-AGENT REINFORCEMENT LEARNING

2.6.2.1 THE STANDARD REINFORCEMENT LEARNING MODEL

Reinforcement learning has been successfully applied in the field of artificial intelligence as well as in operations research. All reinforcement learning methods share the same goal: to solve sequential decision tasks through trial and error interactions with the environment [32]. Every action will invoke a reward granted to the agent.

An open definition summarizing the requirements for an agent is given in [33]: “an agent is a computer system, situated in some environment, that is capable of flexible autonomous action in this environment in order to meet its design objectives”. An agent needs to be reactive and responsive to a dynamic environment, pro-active so as to reach a long-term goal and have the social ability to interact with other agents in a **multi-agent system**. Besides reinforcement learning, planning and supervised learning can also be used to design agents when after selecting an action the correct option is given. RL is a very suitable flexible approach for the design of intelligent agents when planning nor supervised learning are practical. Sometimes of course, as experience is accumulated, an action model can be stated that is progressively based on planning [34].

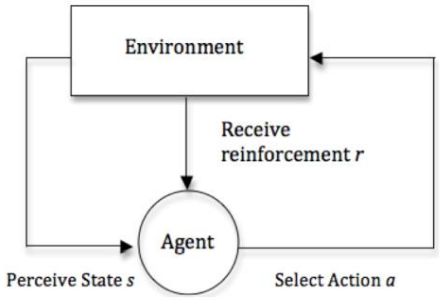

Figure 7. The standard reinforcement learning model [10]

The standard reinforcement learning model comprises:

- a set of states S;
- a set of actions A;
- a set of scalar rewards;
- a transition function T.

At a given instance t, the agent observes a state $s_t \in S$ and linked to a set of actions $A(s_t)$. A reward r_{t+1} is assigned after selecting action $a \in A(s_t)$, and the environment is updated to status s_{t+1}. The probability of selecting an action a at instance t is called the agent’s policy and denoted as $\pi_t(s,a)$. To learn its behavior, an agent requires reward feedback: a reward function determines the immediate reward, while a value function indicates the contribution to the long-term goal [25]. The solution to a RL problem can be obtained by following either a model based approach, using an explicit model of the environment, or otherwise by deriving the optimal policy in a model free approach. Applying the Dynamic Programming technique for instance requires a perfect model of the environment.

2.6.2.2 MULTI-AGENT SYSTEMS

Multi-agent systems (MAS) are more complex than a one-agent system, nevertheless providing a multitude of advantages [26]:

° computational efficiency: concurrent computation;
° extensibility: the number of agents and its behavior can be altered;
° robustness: uncertainty is reduced when agents can communicate, and if one agent fails several other agents remain;
° maintainability: modular composition is easier manipulated than one centralized supra-system;
° responsiveness: again thanks to the modular structure;
° flexibility: agents can be adapted as desired;
° reuse: using the same agent capabilities in different problems.

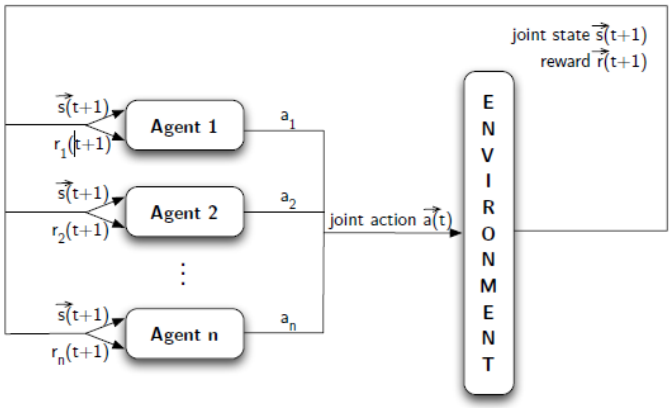

Figure 8. Multiple agents acting in the same environment [25]

The agents can be designed to cooperate or to compete, as illustrated in figure 9.

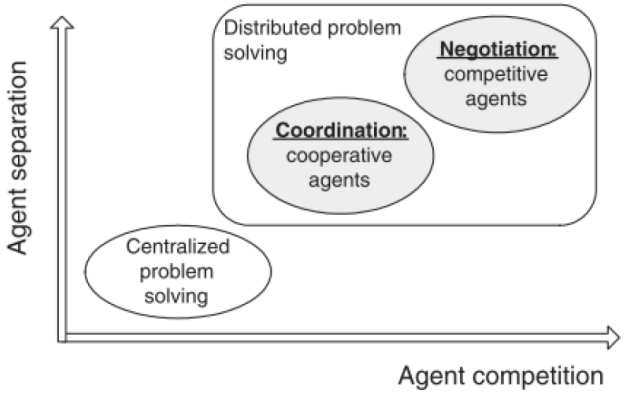

Figure 9. Multi-agent cooperation strategies [25]

2.6.2.3 Q - LEARNING

A popular reinforcement technique is Q-Learning. Each (s,a) pair is associated with a Q-value that is updated according to ([36]):

$$Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s,a)]$$

Notation: $\alpha \in]0,1]$ = learning rate; r is the immediate reward; $\gamma \in [0,1]$ the discount factor, which indicates the present value of future rewards. QL converges to the optimal result under some restrictions.

In RL, an ongoing conflict remains between exploration and exploitation. In order to be successful, the agents have to learn new actions and consider the degree to which they pay off. Concurrently this knowledge must be utilized, yet merely exploring or exploiting doesn't lead to the optimal policy as Sutton & Barto showed in 1998 [32]. A compromis can be made by making the paramaters for the action selection time-dependent, so that in the beginning the agent will explore more, and with time letting the agent exploit more.

2.6.2.4 ACTIONS SELECTION

Three action selection mechanisms are considered. A greedy action strategy always selects the best among the possible actions. The variance of reward may nonetheless lead to a sub-optimal schedule.

A variation is given by the ε-greedy strategy. In (100 - ε) percent of the cases, the best action is selected. For the remaining ε percent, an action is selected at random. A new issues arises, when all actions have the same probability to be selected at random.

The softmax strategy employs the Boltzmann distribution to determine how greedily an agent will operate. Similar to the ε-greedy strategy, one parameter suffices: the temperature τ. The lower the temperature, the greedier the agent.

$$Pr(a) = \frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{m} e^{Q_t(b)/\tau}}$$

2.6.3 EVOLUTIONARY ALGORITHMS FOR RL

The concept of selection and mutation can be combined with RL in an Evolutionary Algorithm for Reinforcement Learning (EARL): the higher the fitness of a population member, the more offspring it will have in the next generation.

Strengths of EARL are the effectiveness in large state spaces, with incomplete information and in non-stationary environments [37]. There are also some important limitations to an EARL. A first limitation is the experimental risk, since the evaluation of a large population is likely to ask for a lot of experiences. Letting an agent freely experiment in a real-world environment may pose a threat or provoke high costs. Secondly, in EA methods no information about bad decisions is withheld, and mutations in an EA can cause the loss of information about a rarely encountered state, even after learning the correct action for such a state.

2.7 PROBLEM DESCRIPTION

2.7.1 EXPECTED SCHEDULING RESULTS AND OBJECTIVE FUNCTION

We already stressed that the floor plan of the Flexible Focused Gearshop (FFG) follows the Flexible Job Shop design. The planning horizon for jobs in the FFG at VCST extends over several months. Once the list of customer orders is available for planning, we also know for each work station which operations are assigned to that particular work station, based on the different processes that the job must go through.

We want the model to determine for every operation on which machine of the work station the operation will be executed, and for every machine which will be the sequencing order and start and finish times of the assigned operations.

Two criteria are of importance. Since VCST is a second-tier supplier, meaning that it supplies the suppliers of the company that does the final assembly, the finishing date may in no case exceed the due date as agreed with the client. This means that the **tardiness** must be zero, if not a huge cost is incurred. Secondly, VCST wants to minimize the **makespan** to cut machine idle time, so that machines and operators can be assigned to new tasks later on.

2.7.2 JOB SPECIFICATIONS & MACHINE SPECIFICATIONS

Following specifications were used for modeling the problem, based on the information provided by VCST:

Job specifications

- A job consists of multiple tasks, each task associated to one of the work stations (a work station is a functional cluster of machines);
- Each operation involves one machine only (as opposed to multi-processor tasks that must be performed on multiple machines);
- A task cannot be interrupted: once the processing of an operation is started, it must be continued until completion. On Friday night, machines can be shut down and the work can be continued straight away on Monday morning;
- The work content of a task cannot be divided over different machines;
- The deterministic processing times are estimated from experience;
- Before the actual processing can start, a machine set-up time and a quality trial run time are taken into account. During the quality trial time, an operator verifies the machine configuration that was adapted during the set-up;
- Each job has a release date, which is the earliest start possible of the processing of the first operation of the job;
- A job has a weight that expresses the relative importance. This weight is used as a multiplication factor in the calculation of the objective value associated to the job;
- Transportation times from and to the different workstations are not taken into account.

Machine specifications

- Machines are clustered according to their functionality to form work stations;
- Only one operation can be processed at the same time on a machine;
- A machine can only perform one type of operation: machines are single-purpose (as opposed to multi-purpose machines that can perform different types of operations);
- Between operations belonging to two different jobs, a change-over must be done. A set-up time is taken into account. The set-up time depends on the machines but not on the sequence of the operations;
- Machines can break down, and some idle time is inevitable. To take into account this efficiency loss, the duration of an operation on any machine is calculated as the deterministic duration divided by the Overall Equipment Effectiveness [38]:

  OEE = Loading x Availability x Performance

  with Loading = scheduled time / calendar time

  Availability = available time / scheduled time

  Performance = (parts produced x ideal cycle time) / available time

- No maintenance activities are included.

CHAPTER 3. MULTI-AGENT REINFORCEMENT LEARNING

3.1 CORNERSTONES OF THE ALGORITHM

3.1.1 CLASSES AND OBJECTS

The algorithm must provide an answer to which product batches will be processed on which machine, in which sequence and at which time. To model the flexible job shop problem, several classes were defined to create objects as workstations, machines, jobs, operations, and actions.

The **job** is the set of tasks, henceforth termed operations, that the product batch must undergo. The dataset provided by VCST contains data about 34 different jobs, hence 34 different product batches that will flow through the flexible job shop department. A corresponding operation is defined for each work station that the job flows through. Each job has an ID (*job_id*), a *lotsize*, the release date starting from which an operation can be planned on a machine (*release_date*), a due date at which the job must be finished (*due_date*), a *weight* to mark a level of importance and of course, the operations comprising the job, stored in a vector (*vec_operations*). The order of appearance in the vector of operations reflects the precedence relations between the different operations of the job.

For each **operation**, the *start* and *finish* times are stored as attributes. An operation has an id (*operation_id*). For the VCST case, the 8-digit operation ID is composed of the six digits of the job ID to which the operations belong, and two more digits to set the ranking number (e.g. the third operation of job 299417 is termed 29941703). The attribute *assigned* is used to make an initial assignment of the operations to one of the qualified machines. The vector of qualified machines (*vec_qualmach*) holds pairs of numbers: the first number in a pair refers to a machine, the second number represents a duration. This means that if the operation would be executed on the machine indicated by the machine number, then the duration would be as given by the second number. Not every single machine at a workstation is necessarily qualified, and the duration of the operation can vary with the machine. To assess the impact of changes in the planning of an operation, the attributes *new_start* and *new_finish* were added.

A **work station** is a functional group of machines that execute the same task. However, not every machine of a work station can execute every task, since there are some physical constraints (e.g. the inner diameter size of a gear might be large so that the gear cannot be held by a machine with small clamps). A work station has two attributes: the workstation ID (*ws_id*) which is a number between 1 and 12 for the VCST case and a vector of machines (*vec_machines*) that contains all the machines located at this workstation.

A **machine** belongs to a work station. The *number* attribute varies from 1 to 33 for the VCST case. Each machine has an overall equipment effectiveness (*oee*), a task-independent set-up time (*setup*) and a task-independent time before the release of an operation by the quality department (*q_trial*). The tasks, which are defined as operations, that will be executed on the machine are listed in a vector of operations (*vec_op_occupation*). To assess the impact of changes made in this task vector, an additional vector was defined (*vec_op_new_occupation*). Only after confirmation that a change is feasible and improves the schedule, the actual vector of operations will adopt the sequence from the temporary vector vec_op_new_occupation, that then will be erased as a preparation for a next iteration. The integration of workstations, machines, jobs and operations is graphically represented in figure 10.

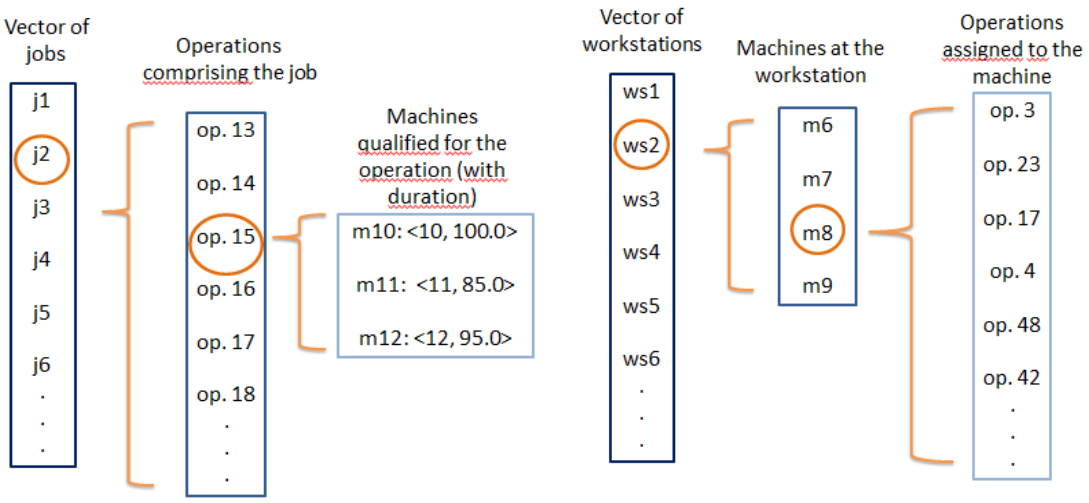

Figure 10. Jobs, operations, work stations and machines

Finally, an **action** has an action ID (*action_id*) which can be **DN** for 'Do Nothing', **IT** for 'insert Idle Time', **RS** for 'Re-Sequence' and **DO** for 'Delegate an Operation'. Every action also has the attribute *objective* which expresses the objective function value as if the action would be executed, and a *ws_count* to refer to the agent. To every workstation one agent is assigned. If upon evaluation the action proves to be feasible, then it will be added to the state of the agent linked to the workstation as specified by the action's ws_count. These general attributes aside, the class action also has action-specific attributes. The *machine_count and IT_position* refer to the position on a machine on which idle time will be inserted for an action with action ID equal to IT. For RS, *RS_it_old* and *RS_IT_new* are used to indicate the old and new position of an operation in the processing order of operations for a machine, along with machine_count that was already defined for IT. For delegating an operation (DO), the numbers *mach_count_del* and *mach_count_rec* refer to the delegating machine and the receiving machine, while *pos_del_op* and *pos_rec_op* refer to the operation on which position in the sequence of the delegating

machine an action will be removed, to be inserted on the specified position in the sequence of the receiving machine. If for a certain action some of the parameters are not needed, then their value is set at -1.

3.1.2 METHODS

Functions that are frequently needed are added in a method, that can be called from anywhere in the algorithm. This allows to keep the code more compact and clear.

- The method ***action_feasible*** verifies for the generated actions which are actually feasible;
- some very useful methods are methods that return pointers to objects, the so-called 'getters'. ***get_job*** and ***get_machine*** return the pointer to the requested object;
- ***job_sort_release*** sorts the jobs according to the earliest release date;
- ***objective*** contains the functions to evaluate the value of the objective function;
- ***print_occupation*** prints an overview per work station of which operations are processed on which machine, with corresponding start and finish times;
- ***rec_mach_qualified*** applies in case the generated action is delegating an operation, and verifies whether the receiving machine is listed in the operation's vector of qualified machines (vec_qualmach);
- ***sort_ascending*** returns a vector of actions that are sorted according to increasing objective value.

The composition of the objective function will be discussed further on this chapter. The objective function is crucial, since the objective function determines the learning behavior and the trajectory towards the final solution.

3.1.3 READING & WRITING DATA

The data for the jobs and machines were delivered in matrix format in the Excel file *(VCST – InputData.xlsx)*. This might be useful and easy to work with to collect the data and represent them to the committee at VCST, but it is not straightforward to read the data directly from the Excel file. To import these data in the heuristic model, we preferably make a detour and rearrange the data in an easy-to-read layout in a text file (*inputData.txt*). Both files are included on a USB key.

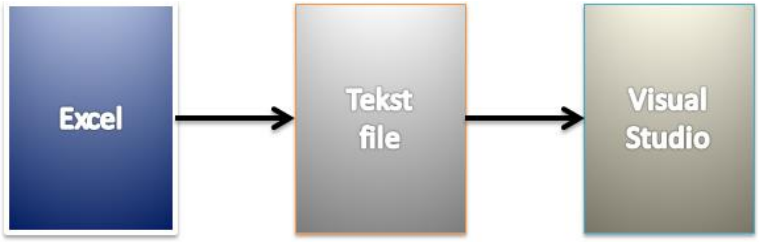

Figure 11. The transformation of input data

An extract from the input data was added in Appendix C. First, the number of jobs, work stations and machines are declared. Then for each of the machines the OEE (as a percentage), the set-up (minutes) and quality trial time (minutes) is given. The next data section shows the assignment of machines (represented by their machine number) to work stations. The final and largest part of the data file contains the information about the jobs. Are represented consecutively for each job: job id, number of operations in the job, lot size, priority weight, release date (minutes), due date (minutes), and a set of operations, one operation per line. An operation is represented by a set of elements consisting of 2 numbers: the first one refers to a machine, the second one is the duration if the operation would be performed on that machine.

The output is written to another text file (*results.txt*) and can be seen directly from within Visual Studio.

3.2 THE FIRST STAGE OF THE AGORITHM

An initial assignment of each operation of every job is made in the first stage, while the second stage contains the actual algorithm with the main loop for gradually improving the initial schedule. To guarantee the randomness in each iteration from the beginning, an operation is initially assigned to a machine that is randomly selected from the set of machines that are qualified for that operation.

3.2.1 QUALIFIED MACHINES & PRECEDENCE RELATIONS

Two types of constraints are considered in this model:

1 Operations can only be performed by **qualified machines**;

2 **Precedence relations** between the operations

2a Precedence between the operations of the same job:
for every job, the order of appearance of operations in the job's vector of operations (*vec_operations*) reflects the precedence relations. The flow of a product batch can be represented by a chain, in which no parallel operations occur.

2b Scheduling operations on a machine:
the next operation can only be processed on the same machine upon completion of the preceding operation. This is furthermore complicated by the set-up time and quality release time, as shown in figure 12.

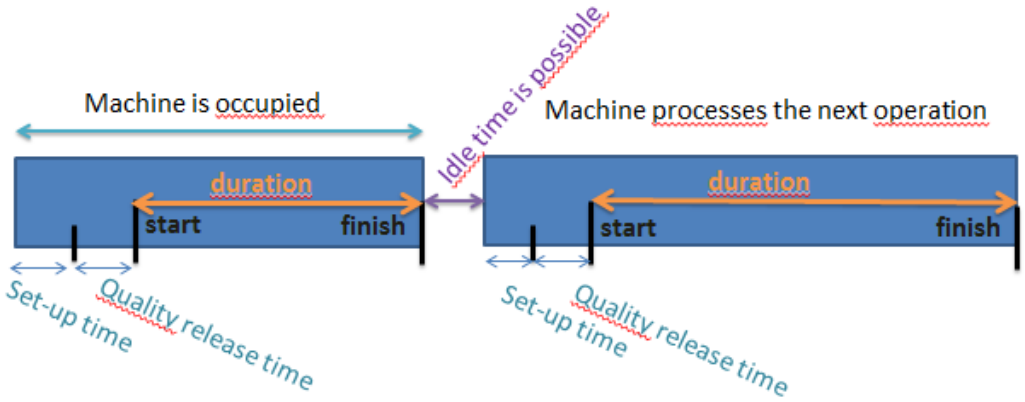

Figure 12. Precedence relations for machine occupation

3.2.2 THE INITIAL ASSIGNMENT OF OPERATIONS

After declaring the various classes and objects that are needed, we can start assigning the operations to a machine. This is done job by job, and per job the operations are assigned one by one with regard to the restriction that only a qualified machine at the appropriate workstation can perform the operation. For each operation, one machine from the set of qualified machines is chosen at random.

If we want to minimize the tardiness and makespan as well, we need to construct an ASAP (as soon as possible) schedule. To set the **start time** of an operation, we take into account the two types of precedence relations resulting in four different scenarios:

- first operation of a job, no preceding operations on the machine:
  start = release time of the job + machine set-up + quality release time
- preceding operations in the job, no preceding operations on the machine:
  start = finish of the preceding operation in the job + machine set-up + quality release time
- first operation of a job, preceding operations on the machine:
  start = max[release time of the job, finish of preceding operation on machine] + machine set-up + quality release time
- preceding operations in the job, preceding operations on the machine:
  start = max[finish of preceding operation in the job, finish of preceding operation on machine] + machine set-up + quality release time

The **finish time** for an activity depends on the availability of the machines reduced by machine breakdowns:

finish = start + (deterministic duration / machine-dependent OEE)

There are some exceptions that must be taken into account for the VCST case. The assignment as described above holds for operations that are performed on workstations 1 to 5 and 7 till 11. However, workstation 6 and 12 are dummy workstations. As described above in the introduction to the VCST flexible Focused Gear Shop, the product batches follow a 3-phase trajectory at VCST. First the product batches undergo some soft activities. Some product batches skip the soft treatment and move directly to the next step, which is being heated at the neighboring Surface Treatment Company (STC). This heat treatment was modeled as a dummy workstation, meaning that an infinite number of jobs can be processed simultaneously. Hence, the precedence relations *2b* don't apply. The start time at STC for a product batch is simply the finish time of the preceding operation in the job, or the job's release date when the treatment

at STC is the first operation of the job. The product batches come back after spending a fixed amount of time at STC, and pass through the third phase at VCST: the hard activities. The result is that a job flowing through workstation 6 experiences a delay. The same goes for workstation 12: the product batch undergoes a final treatment at another company, which can also be modeled as a delay.

3.3 THE SECOND STAGE OF THE AGORITHM

The second stage contains the core of the algorithm: this is where the agents come into action. Per iteration, first all actions are generated. Every workstation is highlighted consecutively to evaluate three different action types, and if the action is feasible it then is added to the state of the agent (a vector of feasible actions) corresponding to the highlighted workstation. For each agent, the best action is selected and put in the first position of the agent's state. Then a ranking of the agents is made to favor the agents whose best action holds the most potential to improve the overall objective. Then, the selected actions are executed, after verifying whether they are still possible. This process is repeated until either the maximum number of iterations is reached, or the solution doesn't improve any further.

In the standard reinforcement learning model, an agent's state is perceived by observing the environment, after which an action is selected and the agent is then rewarded. The environment consists of the occupation of the different machines at the workstation by the (sequence of the) various operations, with their start and finish times that are assigned to these machines. The agent's state is composed by the actions that are feasible at that instance of the algorithm. The action selection is based on the objective value of the actions from the state of the agents.

The agents have no memory, as they do not learn from previously selected actions. They act on an autonomous basis: the agents are non-cooperative. The agents are competitive in as far as their proposed actions are mutually compared and for every iteration a hierarchy is installed, yet they all are dedicated to a common goal which is minimizing the overall objective value as calculated over all jobs.

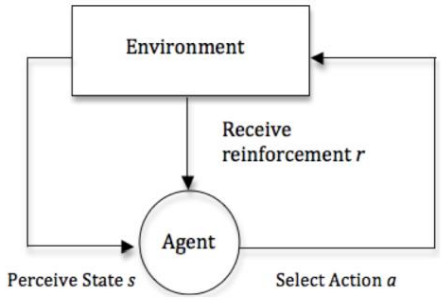

Figure 13. The standard reinforcement learning model (bis)

3.3.1 GENERATING FEASIBLE ACTIONS

Do nothing (DN)

The interpretation of the first type of action is straight-forward. To keep the status-quo, an agent can choose not to perform any action and preserve the environment as observed by the agent. The action DN will be 'performed' when all other actions in an agent's state result in a higher objective value, hence result in a – locally – worse solution. For agent 6 and agent 12, corresponding to the workstations STC – surface treatment and the external Flexible Focused Gearshop, this will be the only action in their state.

Introduce idle time (IT)

The initial idea behind the second type of actions, introducing idle time, is to escape from a local optimal but global sub-optimal solution by creating space between activities. If the gap between two activities on a machine is made wide enough by adding extra idle time, then another operation could be inserted to fill this gap. However, this process turns out to be very inefficient concerning the running time and more important in terms of the quality of the (intermediate) solution.

The combination of the actions RS and DO (see below), supported by secondary mechanisms that influence the action selection but not the action generation, is more appropriate to jump over a local obstacle towards an optimal global solution. Therefore, the action IT was excluded from the algorithm.

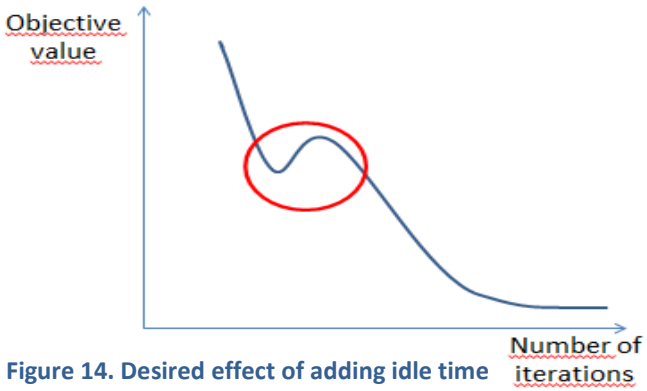

Figure 14. Desired effect of adding idle time

Re-sequence the processing order (RS)

Another type of actions is to change a machine's operations sequence. For every workstation, the algorithm iterates over the different machines at the workstation. For a given machine, if more than 1 operation is at that instance assigned to the machine, then an action is proposed for every operation to be inserted at a different position in the sequence of the same machine.

Let the action RS further be denoted by (it_old, it_new). In the algorithm, the operation at position it_old is first removed from the sequence and then inserted again at the new position. It is not necessary to evaluate every combination of the old position and new position, though. We can already exclude the combinations for which it_old = it_new, since this action is equivalent to the action DN (do nothing). Furhtermore, the effect of action (0,1) is the same as for action (1,0). The same goes for (1,2) compared to (2,1), and (2,3) compared to (3,2). This is not surprising, since the pairs involved represent a pairwise interchange for neighbours, which obviously results in the same final sequence regardless of which of the two positions was the old one, and which one was the new position. These restrictions are summarized by the condition:

$$it\_old \neq it\_new \;\&\&\; it\_new \neq (it\_old - 1)$$

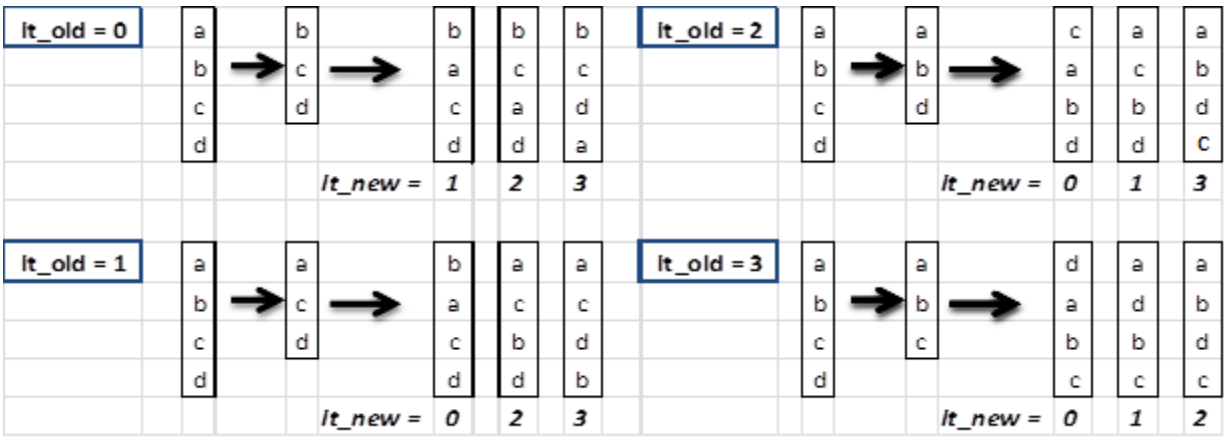

Figure 15. Old and new operation position for generating RS actions

The RS actions are generated starting from the machine's actual vector of occupation (vec_op_occupation). If the condition stated above is satisfied, the machine's new sequence of operations is stored in a second vector (vec_op_new_occupation). Next the start and finish times of the impacted operations is adapted, with respect to the two types of precedence relations for the machine occupation and between the operations of a job. We only need to adapt the times for the operations starting from the position min(it_old, it_new) since the

operations at the lower positions remain unaltered. The new start and finish times are stored under *new_start* and *new_finish*. This completes the necessary action information.

To check the feasibility of the action, we check for the - relevant - operations in the sequence of the machine whether the start of the succeeding operation in the job (precedence type *2a*) doesn't interfere with the new finish of the re-sequenced operation. On the other hand, the set-up of the machine for the successor can start already even if the re-sequenced operation is not yet complete. Also, the finish time of the operation preceding the re-sequenced operation must not interfere with the new start time of the re-sequenced operation.

Hence, the action re-sequencing an operation that has predecessors and successors in the job (precedence 2a) is feasible if it satisfies these two constraints:

(start successor – quality release of successor) >= *finish of re-sequenced operation*
(start of re-sequenced operation – quality release of re-sequenced operation)
>= *finish predecessor*

Quality release represents the time before the quality department allows the actual operation to start. In case the operation considered is the first operation of a job, only the first constraint applies. For the last operation of a job, only the second constraint applies. The constraints can easily be read from figure 12. If the action is feasible, it is added to the state of the agent corresponding the workstation to which the machine, on which an operation is re-sequenced, belongs.

For the VCST case, the dummy workstation 6 and 12 are excluded as we argued before, since their only function is to model a delay in the schedule of a job's operations start and finish times.

Note: The precedence relations 2b demanding that operations on a machine do not overlap need not to be checked since they are already satisfied by construction. The procedure to determine the new start and finish times is similar to set the start and finish times for the operations in the initial assignment. We refer to section 3.2.2 for the detailed description.

Delegate an operation (DO)

The last type of actions is to delegate an operation from one machine to another machine at the same workstation. This is only possible of course if there are multiple machines at the workstation. Again, workstations 6 and 12 are excluded. A machine is selected as the delegating machine, on the condition that there is at least one operation in the machine's occupation since it would be impossible to delegate an operation if none has been assigned. Consequently all

other machines are selected as receiving machines. Then the next machine is selected as delegating machine, and so on for all the machines at the workstation.

Once a machine is selected as the delegating machine, all possibilities are generated for the position of the operation that will be delegated from the position *pos_del_op* in the occupation of the delegating machine. For a given *pos_del_op*, the operation can be inserted at every possible position *pos_rec_op* in the occupation of the receiving machine. Each combination of four numbers (delegating machine, position in delegating machine's occupation sequence *pos_del_op*; receiving machine, position in receiving machine's occupation sequence) represents a single action.

Similar to the RS actions, the impact of executing DO actions is evaluated indirectly by storing the new sequence of actions in a separate vector (*vec_op_new_occupation*). The same goes for setting the new start and finish times for the operations starting from the position *pos_del_op* for the delegating machine, and position *pos_rec_op* for the receiving machine. The start and finish times that come earlier in the occupational sequence remain unchanged. The new start and finish times are again stored under *new_start* and *new_finish*.
To inspect the feasibility of the action, the function action_DO_feasible makes use of the already defined function action_RS_feasible twice, both for the delegating and the receiving machine, to verify if the precedence relations between the operations of a job (precedence type *2a*) are respected. The DO action is feasible only if both the precedence relations for the altered occupation sequence of the delegating machine as well as of the receiving machine are respected. If the action is feasible, it is added to the state of the agent corresponding the workstation to which both the delegating and receiving machine belong.

3.3.2 ACTION SELECTION

The action selection is done in two steps. First, the best action is selected for all agents individually. Then, a ranking of the agents is made again according to ascending objective values of the best actions as proposed by each agent.

The method **sort_ascending** takes as input arguments an agent state, which is a vector of feasible actions that can be performed at the agent's workstation, and returns the agent state sorted according to increasing objective value. For the minimization problem, the best actions have the lowest objective value, and occupy the first positions in the agent state.
A **subroutine** was added **to provide flexibility** to the rigid greedy ranking mechanism. If the first *k* actions in the state of an agent have the same minimum objective value, the sequence of the first *k* actions in the state of the agent is shuffled randomly. Without this subroutine, the

algorithm would get stuck quickly with a sub-optimal solution. In the end, the action that the agent selects as his best shot to be compared to the best actions of the other (competing) agents, is the first action from the vector of actions as returned by the *sort_ascending* method. This is done for every agent (except for the dummy agents linked to workstations 6 and 12).

The second step of the action selection is to make a ranking of the agents based on the objective value of their best action. We simply add the one best action for each agent to a vector. This vector is used as input for the *sort_ascending* method which returns the required ranking. The agents that propose the supreme best actions are found at the first positions in this final vector of best actions (*best_actions*).

Adding the subroutine to *sort_ascending* was one way to add flexibility to the algorithm. Another way to escape from sub-optimal solutions was to adopt the **ε-greedy strategy**. We still start from the agent states that are sorted by the method *sort_ascending*. Now in ε % of the cases the ranking of the best actions of the agents is made at random, regardless of the objective value of the actions. This implies that the selected actions are not the best ones possible in terms of the objective value. On the other hand, since the ranking is made starting from the best actions of the agents the objective value will still improve. In other words, even if the exact trajectory is not optimal there is a trend that pushes the solution in the right direction. What counts is that randomness was introduced. This mechanism is especially relevant when not all agents are allowed to perform their best action, but for example only the first three agents. A large ε means that the algorithm becomes more exploratory by following a less predictable trajectory.

3.3.3 ACTION EXECUTION

After doing all the preparatory work, we can now improve the solution by executing the actions that were selected. The action DN can be straightly executed since it keeps the status-quo. For the actions RS and DO, we must first verify whether the actions are still feasible: due to the precedence relations, the actions of an agent affect the environment from which the actions for the other agents were generated. An example will illustrate this point. Suppose that for a given iteration the first action to be executed is re-sequencing the first operation of a job on a machine at workstation 1 from the first to the last position in the machine's sequence. Suppose furthermore that the second operation is delegating the job's second operation – i.e. the immediate successor of the operation involved in the first action – at workstation 2 from the last position in the sequence of the delegating machine to the first position in the sequence of the receiving machine. The result is that if both actions are executed successively, then the

finish of the job's first operation at workstation 1 might be later than the start time of the job's second operation at workstation 2. This example illustrates the need for verifying the feasibility before the actual execution of an action. Other examples are also possible of course.

The procedure for the execution of the RS and DO actions is completely similar to the procedure to generate these actions at first; only now the new sequence of the operations assigned to a machine is written to the actual vector of occupation (*vec_occupation*) and the real *start* and *finish* attributes are updated.

3.3.4 THE OBJECTIVE FUNCTION

Building an efficient algorithm is only one part of solving the case. Since we are dealing in this algorithm with dumb agents, i.e. they have no short-term memory, the agent can only execute what he has been told – let's say by the director, who determines the operational strategy in the field. This role is played by the objective function, that holds the key to a good-quality solution.

As mentioned before, setting the **tardiness** equal to zero and minimizing the **makespan** are the primary and secondary goals of the optimization problem. To achieve this goal, different approaches are possible based on the same rationale but adopted in different ways.

A total of eight different objective functions were included. For the minimization problem, these partial functions can be summed and weighed by different coefficients. Linear as well as non-linear combinations can then be tested to study the convergence to the final solution, and the quality of the solution in terms of makespan and tardiness. We refer to the next chapter for the results. For now, we suffice by providing an overview of these partial objective functions.

Tardiness

Since this is the absolute priority, the tardiness of each job is weighted by a very big coefficient α that overpowers all other objective functions.

$$\sum_{i \text{ in set of jobs}} \alpha * weight(i) * [finish\ last\ operation\ \ of\ job\ (i) - due\ date\ job\ (i)]$$

Makespan

1) Minimizing the makespan, which is having the shortest finishing time of the jobs as possible, can be interpreted as minimizing the time that machines are idle. If the machine idle time is zero, then the schedule is not necessarily optimal (since the processing time of operations depends on the machine that processes the operation), yet it provides a good starting point:

$$\sum_{m\ in\ set\ of\ machines}[start\ of\ the\ first\ operation + \sum_{j=2}^{Jm} start(j) - finish\ (j-1)]$$

with J_m the number of operations that are scheduled on machine m.

2) One can see that idle time earlier on in the occupation schedule of a machine is more harmful than idle time towards the end, since the number of impacted (= delayed) operations diminishes towards the end. The idle time is weighted by the number of succeeding operations:

$$\sum_{m\ in\ set\ of\ machines}\ [Jm * start\ of\ the\ first\ operation + \sum_{j=2}^{Jm}(Jm-(j-1)) * (\ start(j) - finish\ (j-1))\ ]$$

3) The makespan can be determined by simply looking at the finish of the last operation of each job:

$$\sum_{i\ in\ set\ of\ jobs} weight(i) * finish\ \ of\ last\ operation\ (i)$$

4) The maximum makespan might be distorted by an outlier, even if the proposed schedule is close to optimal. The average makespan is more reliable for a large-scale scheduling problem:

$$\frac{1}{N}\sum_{i\ in\ set\ of\ jobs} weight(i) * finish\ \ of\ last\ operation\ (i)$$

where *N* is the number of jobs to be scheduled.

5) An alternative to considering the maximum or average makespan, is to add the total sum of the makespan for each job:

$$\sum_{i\ in\ set\ of\ jobs} weight(i) * finish\ \ of\ last\ operation\ (i)$$

6) Yet another type of function pertains to the balancing of activities. We include the standard deviation of the makespan to prevent outliers to occur, since they distort the (maximum) makespan :

$$\sqrt{\sum_{i\ in\ set\ of\ jobs} weight(i) * [makespan(i) - avg.makespan]^2}$$

Sequence of operations

Until now, we mainly considered the makespan of jobs, and all partial functions above act on the start and finish times. We might as well think of criteria for the machine occupation.

Imagine a sequence for the occupation of a machine in which the last operations of a job are scheduled up front (e.g. job 1- operation 10; 2-8; 3-4; 4-1). If an operation that comes early in the precedence order of a job is scheduled at the end, then its successors will be delayed all together.
We add a punishment for non-ascending operation numbers (10 - 8 - 4 - 1 in the example above):

$$\sum_{m\ in\ set\ of\ machines} \sum_{j=2}^{Jm} [operation\ number\ (j) - operation\ number\ (j-1)]^2]$$

with J_m the number of operations that are scheduled on machine m.

Workload balancing

The processing time is the complement of the machine idle time; together they account for the total available machine time. The last partial objective function relates to balancing the assigned processing time of machines (and *not* the number of operations assigned to the machines):

$$\sqrt{\sum_{m\ in\ set\ of\ machines} [processing\ time(m) - avg.processing\ time]^2}$$

3.3.5 MECHANISMS TO ESCAPE FROM A SUB-OPTIMAL CONFIGURATION

Randomness is key in this improvement algorithm: the algorithm starts from a random assignment of operations to machines, and in case multiple actions have the same minimal

objective value, then one of these actions is selected at random. However, this randomness also gives rise to sub-optimal solutions since a random path through the solution space doesn't necessary lead to the optimal straight away. It might very well happen, as observed during the simulations, that an agent gets stuck in a sub-optimal configuration.
To overcome the barriers that prevent further improvement, the epsilon-greedy strategy was adopted and a recover point implemented.

3.3.5.1 EPSILON – GREEDY STRATEGY

The first mechanism is the introduction of even more randomness, and was presented before as the ε **-greedy strategy**. This means that in ε % of the cases, the agent will select a random action from the agent's state of feasible actions instead of selecting the action with the lowest objective value.

We introduce some parameters, that of course can be modified:

- The initial value of ε, e.g. set at 0,25 ;
- Thresholds for decreasing or increasing ε, e.g. only decrease ε when higher than 0,20 (some randomness remains) and increasing ε only when lower than 0,50 (no total randomness allowed);
- The decline of ε during each iteration, which can be fixed (e.g. -0,003 per iteration or variable (e.g. $-0{,}40 * \frac{\text{iterations\_count}}{\text{nbr of operations in total}}$) ;
- The decrease of ε is paced by the succession of iterations;
- The increase of ε, e.g. 0,10 ;
- The benchmark is defined as the current objective value in an iteration, as given by the action " Do Nothing";
- The benchmark counter, that counts how many times in a row the objective value doesn't decrease for successive iterations. A decrease of the objective value will reset the counter to 0. The increase of ε is triggered by the benchmark counter, when it exceeds a threshold value (e.g. threshold value of 5 iterations).

A random action can only be selected during an iteration if a randomly generated number in that iteration is lower than the value of ε.

We impose an important additional constraint: random selection is allowed only if the minimal objective value of all feasible actions in that iteration is not lower than the benchmark for that iteration. This means that if an opportunity for improvement still exists, we want to use it. Or to

put it differently, the algorithm isn't stuck yet with a sub-optimal solution.

3.3.5.2 RECOVER POINT

The epsilon-greedy strategy is an effective mechanism to jump over small obstacles, in the short run. However, on the middle-term a stronger mechanism might be needed. This compares to a car that passes an obstacle in a street by driving just around it; if the street is too small then the car will have to return and take a different direction.

During the execution of the algorithm, the global minimum objective is tracked. The benchmark (= the objective value of the action "Do Nothing") at a given iteration is used to evaluate whether the solution improves. If the local benchmark is lower than the global minimum objective, then the last iteration provided an overall improvement. In this case, the global minimum is set equal to this local benchmark and the configuration at this new global minimum is saved as recover point, while a counter is reset to 0. If on the other hand the benchmark doesn't sink below the global minimum, a counter goes up by 1. If the counter exceeds the threshold value, then the agents are summoned to go back to the last recover point. The threshold value defines the exploratory limits of the agent: a high value allows a lot of.

This mechanism is a structural element of the algorithm. It also acts as a corrective mechanism in case the agents stray away too far by choosing a series of random actions, since the agents can return to the last back-up point. This long-term memory also guarantees that after finishing the last iteration, the final solution represents the schedule with the lowest objective value found throughout all iterations: if the final benchmark is (slightly) above the global minimum, then the solution will be reset as well.

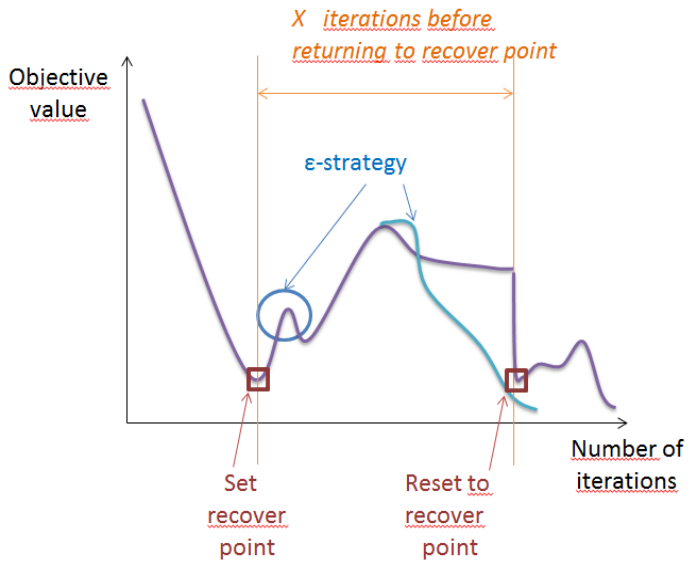

Figure 16. Surpassing a suboptimum through ε-greedy strategy & recover point

3.3.6 PARAMETERS TO TUNE THE ALGORITHM

A family of algorithms can easily be generated by modifying the parameters that affect the behavior of the agents.

The **number of iterations** must be sufficiently larger than the amount of jobs. Suppose for example that every job has an operation at a given workstation. If we want the solution to converge towards a (close-to-)optimal solution, then roughly every operation must have been considered to obtain a new position in the sequence of one of the machines or to be delegated to another machine at the workstation.

The **number of best actions** that are actually **executed** for a given number of iterations also influences the quality of the solution. This is rather obvious for about ε % of the iterations. However, even if we execute actions conform the global ranking of the set composed by the one best action of each agent as based on the objective value, the execution of actions at the end of the ranking might move us away from the optimal.

These parameters concern the running time, as well as the convergence: high values mean that the final global minimum is more likely to be lower, though with repercussions on the intrinsic running time. Another category of parameters concerns the strategy and agility of the agents.

The coefficients and power coefficients of the **partial objective functions** need to be tuned very carefully, since they primarily account for the global strategy throughout each iteration. The objective function is situated at the **strategic level**.

The **value of ε** can be adapted to render the algorithm more exploratory. A trade-off must be made between exploring possibilities and be (locally) less efficient to escape a local sub-optimal solution on one hand, and a solution that converges quickly towards a global optimal solution. The relevant parameters have been already summed up above. This mechanism allows to improve the efficiency locally, on the **operational level** (quick impact in few iterations).

The threshold value for the **recover point** also determines the capability to jump over a local suboptimum on the **tactical level**, in the middle-term (impacts the solution on tens till hundreds of iterations).

3.4 COMPLEXITY OF THE ALGORITHM

The efficiency of the algorithm is measured both in terms of running time and quality of the resulting solution, where the running time intrinsically depends on the complexity of the algorithm. For this algorithm, the complexity is affected by the number of work stations, the number of machines at the work stations and the number of jobs to be scheduled. Let us consider a small example.

Suppose there are *n* jobs to be scheduled on *m* workstations, and each one of the *n* jobs consists of *m* operations which then boils down to one operation assigned to each workstation for each job. So in total, *n* operations need to be scheduled at each of the *m* workstations. Assume furthermore that every workstation counts 3 machines, that are all qualified to perform the operations linked to that workstation.

Consider now an evenly distributed workload, in as far that every machine has been assigned (n/3) operations. This compares to a close-to-convergence state.

- The number of generated DN actions is just m, corresponding to maintaining the status-quo for each workstation.
- The number of generated RS actions is then of the order of mxn^2:

  m workstations *[**3** machines] * [$\frac{n}{3}$ old positions at a machine] *[($\frac{n}{3}$ - 1) new positions at same machine]

  In the beginning however, most of the operations are assigned to one single machine. In case all operations would be assigned to the same machine, the number of generated RS actions is still of the order of mxn^2:

  m workstations *[**1** machine] * [**n** old positions at a machine]*[**n** - 1) new positions at same machine].
- The number of generated DO actions in case of an evenly distributed workload is also of the order of mxn^2 :

  m workstations *3 x [1 delegating machine] * [2 possible receiving machines] * [$\frac{n}{3}$ delegating positions] * [($\frac{n}{3}$ +1) receiving positions].

  When all operations are assigned to one machine, the number of generated actions is of the order of m x n :

  m workstations *[1 delegating machine] * [2 possible receiving machines] * [**n** delegating positions] * [1 receiving position].

  For this configuration, we can conclude that the complexity of the algorithm is directly proportional to the number of workstations and roughly proportional to the square of the total of operations.

However, the actual number generated feasible constraints is subject to more constraints (machines that are qualified, some actions are identical) and more complex than assumed in this example. The number of generated actions will be considered more profoundly in the next chapter, since it influences the running time and so the quality of the solution.

CHAPTER 4. RESULTS

4.1 TEST CASES

4.1.1 ALGORITHM QUALITY AND EFFICIENCY RELATED TO TEST CASES

The quality of the solution has two dimensions: on one hand the gap between the solution found by the algorithm and the upper bound (or if known, the optimal solution), on the other hand the running time to obtain the solution. To assess the quality of the solution proposed by the algorithm, the algorithm is used to solve test cases in section 4.2. The research paper 'Test Instances for the Flexible Job Shop Scheduling Problem with Work Centers' by Behnke and Geiger [39] brings together six sets of common instances for the FJSSP including the best known lower and upper bounds. These datasets are available on the internet. In section 4.3 the algorithm will be applied to the VCST case for which the exact solution is known when the OEE for every machine equals 100%.

The test instances vary in size (number of jobs, number of operations, number of machines and number of qualified machines per operation), as well as in length of the machine-dependent processing times. This leads to the conclusion that not every test instance is representative for the VCST case. This is an important footnote since the running time and hence the efficiency of the algorithm largely depends on the case-specific size and configuration of machines.

In the algorithm, agents can delegate operations from one machine to another at the same work station, implying that all qualified machines for every single operation must belong to the same work station. For the test instances, it follows that all machines are grouped together. This has important implications for the number of generated actions, that rapidly increases with the number of machines. The time required for the generation and evaluation of the feasibility grows correspondingly; furthermore it becomes less likely that the random selection of one of the feasible actions in case of a sub-optimal solution will (quickly) lead to a close-to-optimal solution: the efficiency drops. In any case, a large number of machines at one work station is not representative for the VCST case where at most 5 machines belong to the same work station

The study of the behavior of the algorithm for the smaller test cases allows us to draw some important conclusions, relevant for the analysis of the VCST.

4.1.2 AN OVERVIEW OF THE TEST CASES

Brandimarte introduced the general FJSSP and provided a set of 15 problem instances with medium flexibility. These instances will be the starting point of the evaluation.

Hurink turned some classical JSSP instances into FJSSP instances, by expanding the set of assignable machines for operations according to a particular probability distribution [43]:

edata: *few operations may be assigned to more than one machine*
rdata: *most of the operations may be assigned to some machines*
vdata: *all operations may be assigned to several machines*

For the test instances provided by **Dauzère-Pérès and Paulli**, the number of operations per job is always higher than the number of machines. Hence, a job necessarily has several operations in at least one work station. This is not the case for VCST – at least if we do not consider rework as an activity that needs to be scheduled as well. This set of instances was not tested.

The test instances constructed by **Chambers and Barnes** include at least 11 machines. As mentioned in the previous section, the algorithm is not appropriate for this configuration since all of the machines are grouped together in one work station only. These instances were not tested either.

Kacem et al. designed four instances with total flexibility: each operation can be executed on every machine. These instances also include release dates. Only the data for the first test instance is readily available from [42]. The other instances with 7, 10 and 10 machines are not available, nor representative for the VCST case.

Finally, **Fattahi et al.** introduced twenty small and medium-sized instances that are very appropriate for exact mathematical programming methods, but less for this meta-heuristic.

The instances that were tested are listed in table1 where n is the number of jobs, m the number of machines and $|M_{i,k}|$ the maximum number of assignable machines per operation. An asterisk for the upper bound marks the optimality of this upper bound, in which case it is equal to the lower bound.

Instance	*n*	*m*	$\lvert M_{i,k} \rvert$	*LB*	*UB*
Brandimarte mk01	10	6	3	36	39
Brandimarte mk02	10	6	6	24	26
Brandimarte mk03	15	8	5	204	204*
Brandimarte mk07	20	5	5	133	139
Hurink rdata - mt06	6	6	3	47	47*
Hurink rdata - mt 20	20	5	3	1022	1022*
Hurink rdata - la01	10	5	3	570	571
Hurink rdata - la06	15	5	3	799	799*
Hurink rdata - la11	20	5	3	1071	1071*
Kacem instance 1	4	5	5	11*	11*

Table 1. Selected test cases for the MARL algorithm

Behnke and Geiger also introduce new instances of the FJSSP with work centers in [42]. It is however not possible to measure the performance of the algorithm by solving these instances, since operations can be processed on machines that belong to multiple work stations. Besides, though the design of these instances results from a practice-oriented demand, the conditions and assumptions for these FJSSP with work centers as given in [42] do not apply to the VCST case.

4.1.3 COMPOSITION AND WEIGHTS OF PARTIAL OBJECTIVE FUNCTIONS

Only the maximum makespan is considered for the test instances. Tardiness is not taken into account, which for the VCST case corresponds to jobs having a due date infinitely far away. The methodology used to find the most suitable objective function consists of two steps. In the first step, simulations are run with different weights for the partial objective functions. In the second step, the parameters of the combinations that give good results are tuned.

We already introduced different approaches to minimize the makespan in chapter 3. The test instance Brandimarte mk01 is a good starting point: this problem holds the middle ground compared to the other test instances through its medium flexibility; furthermore the size of the problem is limited, with 10 jobs and 6 machines. In the optimal solution a maximum makespan of 40 is obtained.

The partial objective functions will from now on be referred to by the following code:

A Sum of machine idle time
B Sum of the makespan of every job
C Sequencing
D Workload Balancing
E Weighted sum of machine idle time
F Standard deviation of the makespan
G Maximum makespan of the jobs
H Average makespan over all jobs

Before starting the simulations, we expect that *objective A* makes no distinction between a solution where all idle time is concentrated on one machine, or idle time being spread uniformly over the machines. We also expect it not being able to exploit the machine-dependent process times.
With *objective B* alone, the convergence of the solution is expected to be slow since a lot of actions do not affect the last operation of a job. Hence the makespan of the job remains unaltered even if other actions are needed first to make it feasible to manipulate the job's last operation. This objective function also ignores the occupation of the machines.

Appendix B contains the results for various combinations of the partial functions. The first number is the coefficient, "pow" indicates that the total value of the partial objective function is raised to this power. With a merely experimental approach, these combinations both lead to the lowest maximum makespan of 44:

$$\begin{cases} 0{,}8\ A + 2\sqrt{D} + 0{,}12\ E + G \\ 0{,}8\ A + 2\sqrt{D} + 0{,}12\ E + H \end{cases}$$

We identified the objective functions that are most adequate in obtaining a low makespan, with the values for the secondary parameters in both cases set as follows:

- ε is initially set at 0,15 and decreased by 0,005 per iteration if ε is higher than 0,10; ε is increased by 0,10 if its value is below 0,33 ;
- the benchmark counter is set at 5, meaning that if the objective value of 5 successive iterations doesn't decrease, then ε is increased;
- reset to the recover point after 45 successive iterations of not improving the global minimum;
- running the algorithm for 750 iterations.

The next step is to refine the values of the coefficients and the balance between them. For the remainder of this section, the agents can experiment for 75 iterations before summoned back

to the recover point. The other parameters are unaltered.

Coefficients A & E

The machine idle time is chosen as the basis of the objective function. From the first simulations, it turns out that workload balancing can drastically improve the solution. We simultaneously tune two parameters in the following objective function:

$$coeff_A * A + \sqrt{D} + coeff_E * E$$

Max makespan	Avg makespan	$coeff_A$	$coeff_E$
61	43.6	0.2	0.2
65	46.1	0.2	0.4
78	48.3	0.2	0.6
110	54.7	0.2	0.8
118	59.9	0.4	0.2
61	43.2	0.4	0.4
82	47.3	0.4	0.6
84	54.6	0.4	0.8
48	39.5	0.6	0.2
77	48.5	0.6	0.4
144	80.2	0.6	0.6
50	39.6	0.8	0.05
47	**35.6**	**0.8**	**0.8/8 =0.1**
58	37.4	0.8	0.15
59	39.2	0.8	0.2
121	64.1	0.8	0.4
92	55.1	0.8	0.6
104	66.3	0.8	0.8

Table 2. Results for Brandimarte mk01 with objective A & E

Two aspects are rendered visible:

1 A > E : the position of idle time is important, yet the weight as attributed in the calculation of objective E is too large and must be compensated for.

2 The value of the ratio A/E must be large, e.g. 7 or 8 yet the exact value of the ratio fluctuates. The results are found to be rather sensitive to the exact ratio, and subject to randomness.

Coefficients A & D & E

Of course, the value of objective function D can vary as well. One could see that the coefficient value of 1 for D is too low, since the occupation of the machines is quite out of balance for the previous simulations. This presumption is checked for in several experiments:

Max makespan	Avg makespan	$coeff_A$	$coeff_D$	$coeff_E$
49	37.1	0.8	**1.5; pow 0.5**	0.1
47	36.3	0.8	**1.75; pow 0.5**	0.1
44	35.3	0.8	**2; pow 0.5**	0.1
49	34.9	0.8	**2.12; pow 0.5**	0.1
45	34.3	0.8	**2.25; pow 0.5**	0.1
56	***35.9***	***0.8***	**2.5; pow 0.5**	***0.1***
45	35.1	0.8	**2.5; pow 0.5**	0.1
48	35.6	0.8	**2.75; pow 0.5**	0.1
53	38.9	0.8	**3; pow 0.5**	0.1

Table 3. Results for Brandimarte mk01 with objective A, D & E

Both the maximum makespan and the average makespan decrease to a minimum for $coeff_D = 2,25$. Some simulations give bad results. The possibility that this occurs even with an efffective objective function cannot be excluded since randomness is involved in this improvement algorithm. The general trend for increasing the value of $coeff_D$ is clear, though.

Coefficients for G, for H and for both G & H

Adding the last two objective functions, we obtain following results:

Max makespan	Avg makespan	$coeff_A$	$Coeff_D$	$coeff_E$	$coeff_G$
48	36.1	0.8	2.25; pow 0.5	0.1	0.5
48	34.8	0.8	2.25; pow 0.5	0.1	0.75
48	36	0.8	2.25; pow 0.5	0.1	1
48	39.4	0.8	2.25; pow 0.5	0.1	1.25
50	38.3	0.8	2.25; pow 0.5	0.1	1.4
42	34.4	0.8	2.25; pow 0.5	0.1	1.5
43	35.3	0.8	2.25; pow 0.5	0.1	1.5
47	34.6	0.8	2.25; pow 0.5	0.1	1.5
43	33	0.8	2.25; pow 0.5	0.1	1.5
42	31.8	0.8	2.25; pow 0.5	0.1	1.5
46	34.3	0.8	2.25; pow 0.5	0.1	1.6
53	36.6	0.8	2.25; pow 0.5	0.1	1.6
49	35.6	0.8	2.25; pow 0.5	0.1	1.75
60	42.5	0.8	2.25; pow 0.5	0.1	2
48	35.5	0.8	2.25; pow 0.5	0.1	2
47	34.9	0.8	2.25; pow 0.5	0.1	2.25
48	35.5	0.8	2.25; pow 0.5	0.1	2.5

Table 4. Results for Brandimarte mk01 with objective A, D, E & G

Max makespan	Avg makespan	$coeff_A$	$coeff_D$	$coeff_E$	$coeff_H$
49	35.2	0.8	2.25; pow 0.5	0.1	0.75
48	36.6	0.8	2.25; pow 0.5	0.1	1
49	35.3	0.8	2.25; pow 0.5	0.1	1.25
47	37.7	0.8	2.25; pow 0.5	0.1	1.5
47	32.1	0.8	2.25; pow 0.5	0.1	1.75
48	36	0.8	2.25; pow 0.5	0.1	2
47	34.9	0.8	2.25; pow 0.5	0.1	2.25
49	35	0.8	2.25; pow 0.5	0.1	2.5

Table 5. Results for Brandimarte mk01 with objective A, D, E & H

The best results are obtained by a combination of A, D, E and G, more specifically for $coeff_G$ = 1,5. To make sure that the results are not due to randomness, so that a lucky simulation distorts the view, the simulation was repeated several times. The average of the maximum makespan over 5 identical simulations is 43,4 with a standard deviation 2,07. Adding H to the objective function along with A, D, E and G doesn't lead to a further improvement, on the contrary. Based on the previous findings, the combination of G/H = 8 and G + H =1,5 was also tested, yet without success:

Max makespan	Avg makespan	$coeff_A$	$coeff_D$	$coeff_E$	$coeff_G$	$coeff_H$
57	35.2	0.8	2.25; pow 0.5	0.1	0.75	0.875
45	36.6	0.8	2.25; pow 0.5	0.1	1.125	1.31
48	35.3	0.8	2.25; pow 0.5	0.1	1.5	1.5
44	37.7	0.8	2.25; pow 0.5	0.1	1.5	1.75
47	32.1	0.8	2.25; pow 0.5	0.1	1.5	2
48	36	0.8	2.25; pow 0.5	0.1	1.33	1.33/8 = 0.16

Table 6. Results for Brandimarte mk01 with objective A, D, E, G & H

Based on the test instance Brandimarte mk01 and running simulations for various combinations of coefficients and powers, the objective function that performs best is:

$$0{,}8 * A + 2{,}25 * \sqrt{D} + 0{,}1 * E + 1{,}5 * G$$

The way to get to this objective function is rather intuitive, other objective functions probably could be found that might achieve the same or even better results. To conclude, we can state that with this limited set of simulations we constructed an objective function that for the test instance Brandimarte mk01 achieves good results. To assess the effectiveness of this objective function, the algorithm must be applied to other test instances as well.

4.1.4 RESULTS FOR THE SELECTED TEST CASES

4.1.4.1 BRANDIMARTE INSTANCES

Brandimarte mk01 (10 jobs, 6 machines, UB = 40 = optimal solution)

The objective function is already known, the configuration of parameters for the epsilon-greedy strategy remains the same as well. The number of iterations before resetting to the recoverpoint and the total number of iterations varies:

Max makespan	Avg makespan	Recover point	# iterations	Running time (*)
42	34.4	75 it	750	16 min
45	36.2	100 it	1000	22 min
56	34.7	150 it	1000	21 min
41	36.2	200 it	2000	45 min

Table 7. Results for Brandimarte mk01 with full objective

(*) All test instances were run on an Intel® Core™ i7 Processor 2.30 GHz, and 6 GB RAM

We see again that randomness cannot be excluded: a similar experiment with 1000 iterations but slightly different learning behavior (number of iterations before resetting to recover point) results in a difference in maximum makespan of 11, yet the average makespan remains low. We read from table 7 that for 2000 iterations and recover point at 200 iterations the gap between the algorithm's solution and the optimal solution is 2.5%: increasing the learning rate and the total number of iterations is beneficial in terms of closeness to optimality.

Brandimarte mk02 (10 jobs, 6 machines, UB = 27)

Max make-span	Avg make-span	$coeff_A$	$coeff_D$	$coeff_E$	$coeff_G$	Recover point	# iterations	Running time
42	38.9	0.8	2.25	0,1	1.5	75 it	750	37 min
44	38.1	0.8	2.25	0,1	1.5	75 it	750	40 min
47	39.3	**0.9**	2.25	**0**	1.5	75 it	750	38 min
40	35.3	0.8	2.25	0,1	1.5	100 it	1000	48 min
43	38.3	0.8	2.25	0,1	1.5	100 it	1000	50 min
40	37.8	0.8	2.25	0,1	1.5	150 it	1000	48 min
41	37.5	0.8	2.25	0,1	**2**	150 it	1000	49 min
39	36.7	0.8	2.25	0,1	1.5	200 it	2000	92 min
35	32.9	0.8	2.25	0,1	**2**	200 it	2000	95 min

Table 8. Results for Brandimarte mk02

Again the combination of a high number of total iterations and a high tolerance before resetting to the recover point gave the best results. The minimal gap in makespan is quite big: 35 compared to an upper bound of 27, which is 22.3% more. Compared to mk01, where the gap was only 2.5%, we find that mk02 has twice as much assignable machines per operation, so the solution space grows larger and more re-sequencing and delegation actions are possible. Testing for more iterations is likely to further improve the solution.

Brandimarte mk03 (15 jobs, 8 machines, UB = 204 = optimal solution)

Max makespan	Avg makespan	Recover point	# iterations	Running time
263	217.4	75 it	1000	492 min

Table 9. Results for Brandimarte mk03

The original objective function 0,8 * A + 2,25 * $\sqrt{\mathbf{D}}$ + 0,1 * E + 1,5 * G was used. The running time for 1000 iterations expands to several hours, given that there are now 8 machines at the work station. If one can see from figure 17 and figure 18, for instances with 8 and more machines the convergence is very slow since the random action selection is less likely to pick actions that lead to breakthroughs. We expect to find a better solution by increasing the total number of iterations and with enhanced learning behavior. This instance was not tested in depth, since simulations are time-intensive and not representative for the VCST case due to the excessive number of machines.

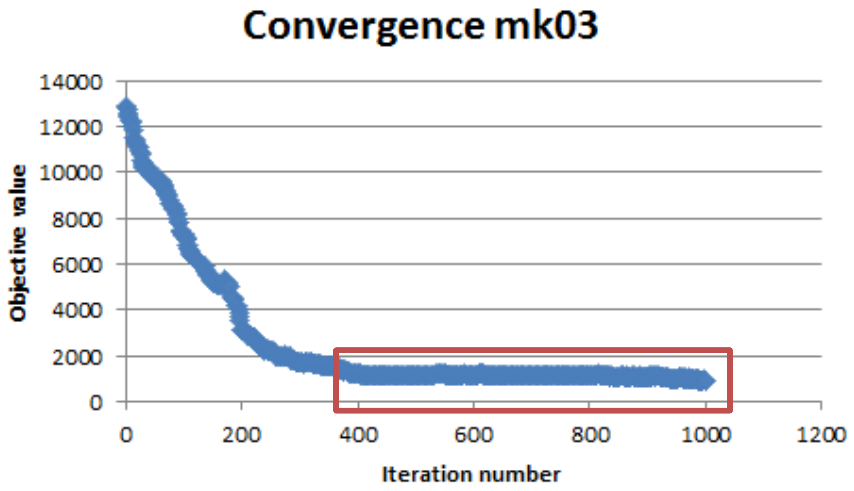

Figure 17. Convergence of test instance Brandimarte mk03 with standard objective function

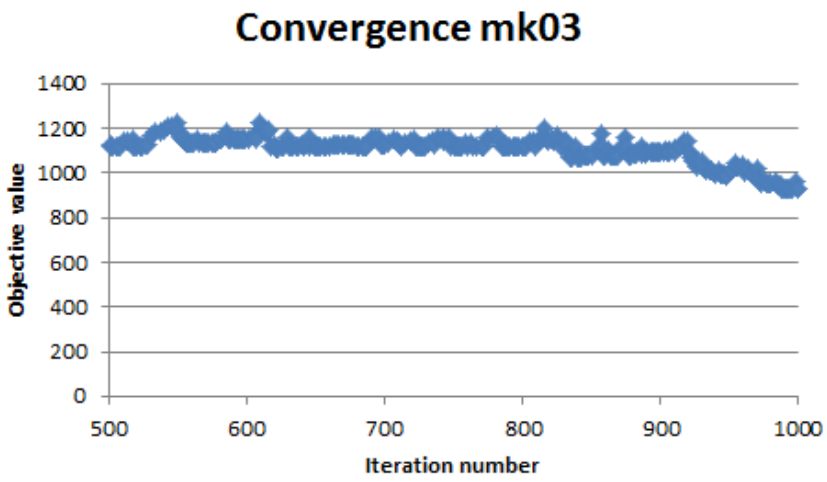

Figure 18. Detail of convergence for test instance Brandimarte mk03

The dimension of time concerns the convergence expressed as a number of iterations, and the duration per iteration. As we can see from figure 17 and 18, up till iteration 400 convergence is fast, after which a slow process of small improvements takes over. After 1000 iterations, an improvement of only 20% is reached compared to the solution after 400 iterations. The duration of one iteration depends on the number and flexibility of machines (i.e. how many operations can a machine process). For the instance mk03, generating (and evaluating) 900 actions per iteration is not uncommon which results in the excessive running time. This is a structural element of applying the meta-heuristic to this specific test instance. Limiting the number of generated actions might reduce the running time. On the other hand, more advanced optimization of the C++ code will most probably considerably shorten the running time as well.

Brandimarte mk07 (20 jobs, 5 machines, UB = 143)

Max makespan	Avg makespan	Recover point	# iterations	RS & DO shortcut	Running time
224	179.55	75 it	750	No	330 min
238	189.05	75 it	500	150 actions	156 min
225	189.45	75 it	800	No	412 min
239	200.85	75 it	1000	300 actions	290 min
223	190.7	150 it	1000	No	334 min

Table 10. Results for Brandimarte mk07 with full objective function

Since the running time for mk07 extends to several hours, a shortcut was created. If after the generation of a minimum number of feasible re-sequencing (and delegating) actions – e.g. 150 – the lowest objective value of those 150 actions is at least 0,5% below the status-quo, then no new re-sequencing actions will be generated (and similar for delegating actions). This saves about 15% in running time over 1000 iterations, at a cost of ending up with a maximum makespan that is higher than when the agent can evaluate every feasible action.

There is no significant difference between the results for mk07. This test instance illustrates an important limitation of the algorithm: the gap of the maximum makespan with the upper bound, in this case 55,9%, is large when the processing times of operations change considerably depending on the machine. The objective function as stated before doesn't take this into account, and hence is ineffective in selecting the right actions. The quality of the solution depends largely on random effects. We did not further exploit this instance, since in the VCST case the processing times are only lightly machine-dependent.

4.1.4.2 HURINK INSTANCES

For the Hurink instances, we chose the set rdata, i.e. most of the operations may be assigned to some machines since this compares best to the VCST case.

Hurink rdata mt06 (6 jobs, 6 machines, UB = 47 = optimal solution)

Max makespan	Avg makespan	Recover point	# iterations	Running time
50	43.33	75 it	1000	7 min
50	43.8	40 it	1000	7 min
52	42.83	100 it	1000	7 min
54	40.5	150 it	2000	13 min

Table 11. Results for Hurink rdata mt06

The lowest maximum makespan found with the MARL algorithm shows a gap of 6.3% compared to the optimal solution.

Hurink rdata mt20 (20 jobs, 5 machines, UB = 1024)

Max makespan	Avg makespan	$coeff_G$	Recover point	# iterations	Running time
1115	832.75	1,5	100 it	1000	158 min
1026	884.15	1,5	200 it	2000	320 min
1041	866.3	2	200 it	2000	314 min

Table 12. Results for Hurink rdata mt20

The lowest makespan out of the three simulations shows a gap of 0.39% with the optimal solution.

Hurink rdata la01 (10 jobs, 5 machines, UB = 573)

Max makespan	Avg makespan	$coeff_G$	Recover point	# iterations	Running time
601	558.9	2	200 it	1000	17.5 min
588	537.6	2	200 it	2000	39 min
618	539.1	1.5	200 it	1000	18 min
613	532.8	1.5	200 it	2000	44.5 min

Table 13. Results for Hurink rdata la01

With the coefficient of component G in the objective function increased to 2, the minimum gap of the maximum makespan is 2.6%.

Hurink rdata la06 (15 jobs, 5 machines, UB = 799 = optimal solution)

Max makespan	Avg makespan	$coeff_G$	Recover point	# iterations	Running time
804	678.2	2	200 it	1000	57 min
810	698.2	2	200 it	2000	116 min
2612	1197.8	1.5	200 it	1000	57 min
801	727.4	1.5	200 it	2000	113 min

Table 14. Results for Hurink rdata la06

The minimum gap of the maximum makespan is 0.25%, which is almost optimal.

Hurink rdata la11 (20 jobs, 5 machines, UB = 1071 = optimal solution)

Max makespan	Avg makespan	$coeff_G$	Recover point	# iterations	Running time
1073	962,65	2	200 it	1000	178 min
1080	930.2	2	200 it	1000	182 min

Table 15. Results for Hurink rdata la11

The first simulation shows a minimum gap of the maximum makespan of 0.19% .

Kacem instance 1 (4 jobs, 5 machines, UB = 11 = optimal solution)

Max makespan	Avg makespan	$coeff_B$	$coeff_D$	$coeff_G$	Recover point	# iterations	Running time
16	14	0	2.25	1.5	200 it	2000	171 sec
17	13	0	2.25	2	200 it	2000	170 sec
12	10	0	2.25	2	200 it	2000	170 sec
18	12.5	0	2.25	2	300 it	2000	168 sec
16	12	0.5	2.25	2	300 it	2000	170 sec
20	13.5	0.75	2.25	2	300 it	2000	169 sec

Table 16. Results for Kacem instance 1

The Kacem instances are characterized by total flexibility: each operation can be processed on each machine. The gap is established at 9.1%. The proposed solution is graphically represented in figure 19.

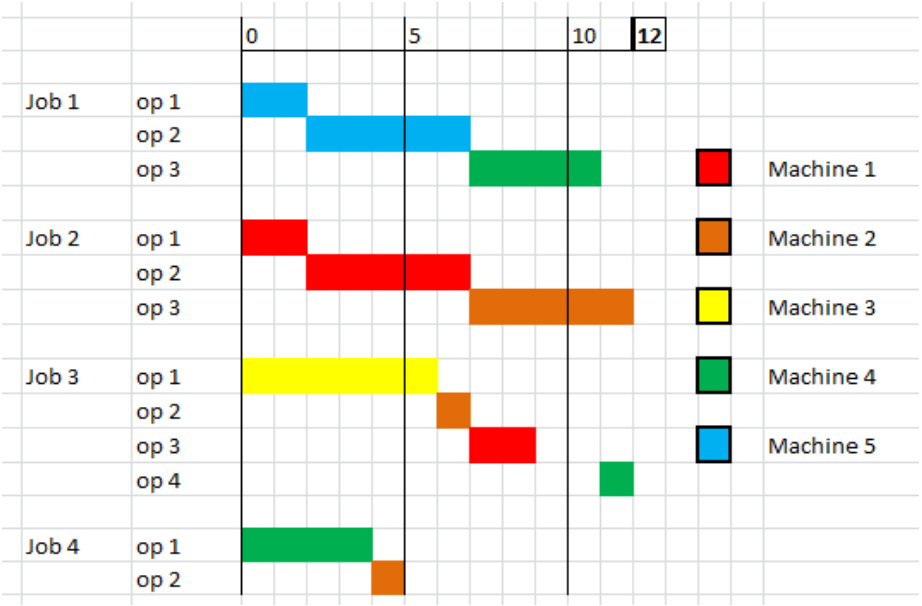

Figure 19. Job schedule for Kacem instance 1 (without release dates)

4.1.5 CONCLUSIONS FOR THE TEST INSTANCES

The results for the test instances are summarized in table 17:

Instance	*n*	*m*	*UB*	*MARL*	*Gap (%)*
Brandimarte mk01	10	6	39	41	2.5
Brandimarte mk02	10	6	26	35	22.3
Brandimarte mk03	15	8	204*	263	28.9
Brandimarte mk07	20	5	139	23	55.9
Hurink rdata - mt06	6	6	47*	50	6.3
Hurink rdata - mt20	20	5	1022*	1026	0.39
Hurink rdata - la01	10	5	571	588	2.6
Hurink rdata - la06	15	5	799*	801	0.25
Hurink rdata - la11	20	5	1071*	1073	0.19
Kacem instance 1	4	5	11*	12	9.1

Table 17. Summary of simulation results for the test instances

The column '*MARL*' contains the best solution obtained with the algorithm proposed in this thesis. The gap expresses the width of the gap with the optimal solution as a percentage. From the test instances, we can conclude:

- With the standard objective function ***0,8 * A + 2,25 * $\sqrt{D}$ + 0,1 * E + (1,5 or 2) * G*** constructed via experiments, we generated results that come close to optimality for instances similar to the VCST case as to the number of machines per work station;
- Learning is an important aspect: by exploring the solution space via random action selection, an improved solution can be found instead of getting stuck with a sub-optimal solution;
- The objective function is best suited to a limited set of machines per work stations and processing times that are only slightly machine-dependent;
- Important limitations on the efficiency of the algorithm are the number of machines at a work station, the difference in processing times depending on the machine to which an operations is assigned, and a high degree of flexibility.

For instances that are similar to the design of the VCST case and in which these limitations do not apply, a schedule is created that comes close to optimality.

4.2 THE VCST CASE

4.2.1 WORKING METHOD

In the VCST case, 34 jobs must be scheduled on 12 work stations with a varying number of machines assigned to the work stations (see Appendix A). An exact solution (makespan = 536,72h) is known from [12] with the important footnote that the OEE for every machine is assumed to be 100%.

This section is structured as follows. In parallel, the objective function parameters for OEE = 100% and for OEE < 100% are tuned in three steps. The weights of the standard objective function are chosen as the default settings for tuning the objective coefficients. The reduction of machine idle time can be argued to be the top priority, which is why parameters $coeff_A$ =0.8 and $coeff_E$ = 0.1 are chosen as benchmark. A and E are complementary, and therefore considered together. First, $coeff_D$ is set at 2.25 and $coeff_G$ varies. Secondly, we let $coeff_D$ vary while $coeff_G$ = g* with g* the value of $coeff_G$ that lead to the best result in the first step, while still $coeff_A$ =0.8 and $coeff_E$ = 0.1. Finally, all weights are known and the "optimal" objective function is tested for its mean and standard deviation over 10 simulations. We note that letting the value of $coeff_A$ (and $coeff_E$) vary is redundant, since only the ratio of the coefficients and not the absolute value is important.

To tune the parameters, the parameter is increased in steps of 0.25 around the default settings. Some configurations perform better, and a pattern should become visible. We then zoom in on the best result for the variable parameter, by halving the interval for this parameter for successive simulations, e. g.

…	x-0.50	x-0.25	***[x]****	x+0.25	x+0.50	…
…	x-0.25	***[x-0.12]****	x	**x+0.12**	x+0.25	…
…	x-0.25	**x-0.18**	x-0.12	**x-0.06**	x+0.12	…

to determine the most opportune objective function.

Obviously, this approach is merely experimental and it is not guaranteed that the quality of the solution will be close to optimal. This approach will allow us to draw some important conclusions.

Ideally, a (fractional) factorial design would indicate the most suitable combination and interaction of the partial objective functions. However, the parameters are multi-level (the

weight can be any positive number), and generating the results for different parameter combinations is rather time-intensive.

We assume that one simulation for a given configuration of parameters is sufficient to evaluate the performance. In other words, if we assume that the results are normally distributed, then we assume that the standard deviation is small enough so that the result for one simulation comes close to the mean that we would find by repeating the simulation several times. This sounds plausible, since the randomness strategies (epsilon, recover point) guarantee that multiple trials are undertaken to overcome local barriers during one simulation. Hence, we assume that the final result depends heavier on the parameters of the objective function than on randomness since the occurrence of sub-optimal solutions is reduced by introducing this randomness. This assumption can only be valid if enough trials are undertaken faced to sub-optimality barriers, and of course depends also on the secondary parameters and the configuration of the optimal solution. This assumption must be verified afterwards. For now, if for one simulation the result differs significantly for parameter values that are close, then this particular simulation will be run again.

4.2.2 RESULTS AND CONSIDERATIONS FOR THE VCST CASE

An overview of the results of the simulations can be found in Appendix D (OEE = 100%) and Appendix E (OEE < 100%). For each combination, the averaged maximum makespan over 10 simulations in step 3 is always higher than the reference level from step 2 of the procedure described in the previous section. If we accept the hypothesis that the maximum makespan is normally distributed for a given objective function, then we can calculate the k-value and the corresponding percentile conform the standard Gaussian distribution.

For OEE = 100%, three combinations were tested:

- $coeff_A = 0.8$ $coeff_D = 3.5$ $coeff_E = 0.1$ $coeff_G = 0.75$
  The **minimum makespan** from step 2 is **675**; in step 3 ten more simulations are executed resulting in an averaged maximum makespan of $\mu = 747$ (gap of 39.1% with the optimal solution) and $\sigma = 56.7$, z-value = -1.26 corresponding to the 10^{th} percentile.
- $coeff_A = 0.8$ $coeff_D = 3.87$ $coeff_E = 0.1$ $coeff_G = 2$
  The **minimum makespan** from step 2 is **678**, we find in step 3 $\mu = 839$ and $\sigma = 92.9$, z-value = -1.73 corresponding to the 4^{th} percentile.

- $coeff_A = 0.8$ $coeff_D = 0.25$ $coeff_E = 0.1$ $coeff_G = 2.37$
  The **minimum makespan** is **593**, with μ = 806 and σ = 48.5, z-value = -4.93, situated on the far left in the 1st percentile.

From [12], the exact solution is known to be 536. The results above are significantly above this exact solution.

For OEE < 100%, two combinations were tested:

- $coeff_A = 0.8$ $coeff_D = 3.5$ $coeff_E = 0.1$ $coeff_G = 0.85$
  The **minimum makespan** from step 2 is **757**, we find in step 3 μ = 883 and σ =55.8, z-value = -2.26 corresponding to the 2nd percentile.
- $coeff_A = 0.8$ $coeff_D = 2.5$ $coeff_E = 0.1$ $coeff_G = 1.75$
  The **minimum makespan** from step 2 is **780**, we find in step 3 μ = 965 and σ = 77.0, z-value = -2.40 corresponding to the 1st percentile.

We want to select the objective function that is most likely to give good results. Suppose that we accept a solution only if it is no more than 10% away from optimality (i.e. makespan of maximum 590h). The probability of finding such a solution is respectively 0.28% , 0.36% and 0.0004% when OEE = 100%; and 0.0001% and 0.00006% in case OEE < 100%. This is clearly not acceptable for the VCST management. This analysis is based on a sample with size 10, which is too small to draw sound conclusions. Additionally, it seems rather unlikely that the maximum makespan follows a normal distribution. We would expect to see a stepwise profile in the cumulative probability density corresponding to the critical chain in the different schedules. However, it seems that the solution with a maximum makespan of 593 has been obtained by lucky coincidence.

Another point of interest is the value of the objective function compared to the makespan. For the selected configurations of objective parameters, only the first from the three combinations for OEE = 100% guarantees that the makespan is monotonously decreasing with decreasing objective value. This implies that for the other 4 selected objective functions, solutions to this minimization problem might be found with a higher objective value yet a lower makespan! There is a conflict, the objective function does not command the agents' behavior in the required way.

We already discussed the variability of the solution and the conflicting behavior related to the selection of the objective function. Another issue arises, precisely related to the variability and conflicting behavior. The coefficient of partial objective function G, which is nothing more than the maximum makespan, reaches a maximum value of 2.5 only in the experimental composition of the objective function. Increasing $coeff_G$ will eliminate the conflicting behavior and might

reduce the variability as well. Similarly, $coeff_D$ reaches a maximum value of 4.5. Maybe a value of $coeff_D = 10$ gives a better result? The only way to determine the upper limit is by running more simulations with increasing values for these parameters, until the solution is clearly moving away from the optimal. Given the variability of the solution, these experiments would have to be repeated several times. More elaborated testing is needed, up to the point that it becomes unworkable given that this is only a test instance.

CHAPTER 5. CONCLUSION & FURTHER RESEARCH

5.1 STRENGHTS AND WEAKNESSES OF THE MARL-ALGORITHM

In this thesis, a Multi-Agent Reinforcement Learning Algorithm is proposed for the Flexible Job Shop Scheduling Problem. A list of jobs, each job consisting of a set of operations, must be scheduled on at least one out of twelve work stations: for each operation, a qualified machine must be selected and the start time of the operation must be determined.

Once the structural elements and secondary mechanisms of the algorithm are swung into place, several partial objective functions were proposed. Then the performance of the algorithm was measured against some test instances that are readily available on the internet for which the upper bounds, and in many cases the optimal solution as well, are known. Using the 'standard' objective function, solutions that are less than 1% from the optimal solution can be obtained for some specific test cases. Important limitations on the efficiency of the algorithm are the number of machines at a work station, the difference in processing times depending on the machine to which an operation is assigned, and a high degree of flexibility.

The strength of the algorithm lies in the decomposition in work stations operated by autonomous agents. As long as the number of machines at a work station is limited to a maximum of about 6, the agent is quite efficient in generating and evaluating actions. Another point of interest is that not every possible solution in the solution space needs to be evaluated to find a (close-to-)optimal solution.

In chapter 4, we discussed the variability and conflicting behavior of the agents, that has its origins in the composition of the objective function. The most important weakness lies in the construction of the objective function. The question that we failed to answer clearly, is how the most opportune objective function can be determined i.e. which partial objective functions should be represented and what are the relative weights? Finding an exact method is not straight-forward; an experimental approach is time-intensive. And if the exact solution of any case is not known beforehand, how can you be sure that the proposed solution is at least close to optimal, when the most suitable objective function is case-dependent?

Finding a method that leads to the construction of a well-performing objective function, based on the configuration of the work stations and the degree of dependence of the machine processing times, would force a breakthrough. As the American psychologist prof. Paul Meehl argued in his 1954 *'Clinical vs. Statistical Prediction: A Theoretical Analysis and a Review of the Evidence'*, mechanical (or meta-heuristic) methods should be favored over professional judgment in the prediction of behavior. A statement that is convincingly supported by Orley

Ashenfelter's formula to predict the quality of a wine, based on three key parameters [40].

5.2 FURTHER RESEARCH OPPORTUNITIES

A topic of future research to improve the quality of the algorithm could be determining such an objective function construction method, or a simple formula that calculates the weights for the partial objective functions. We believe however that a more successful approach would be to deploy smarter and more cooperative agents. Currently, each agents acts completely independently of the others, and the agents have no memory. Mechanisms need to be introduced that can link together the action selection over multiple iterations and for multiple agents as part of a team. This will increase the probability that the right combination of successively executed actions will reduce the makespan, as well as reduce the influence of randomness in the process. To do this, we believe that the identification of the critical chain in the schedule or the bottleneck machines is crucial. This will however elevate the level of ingenuity of the algorithm to the more challenging level in the field of artificial intelligence, requiring an algorithm make-over and advanced programming skills.

A need from reality that was not addressed in this thesis, is taking into account the WIP, as well as the availability of operators. Machine breakdowns on the other hand are considered only indirectly, using the OEE that is known from historical data. Creating the possibility to insert a machine breakdown in a dynamic schedule, would improve the fit with reality.

Some other aspects to improve the applicability and user-friendliness could be building a user-interface that facilitates the input of data; the automatic generation of a Gantt chart based on the final schedule or presenting the data output conform the VCST working calendar with the possibility of inserting extra night or weekend shifts.

For the VCST case, the solution found by the algorithm remains far from optimal. While solutions showing a gap of about 25% - 30% from optimal can be found rather frequently, the probability of obtaining a solution that is at most 10% away from optimal is close to zero. Now, even if a person can obtain the same quality in planning (around 25% away from optimal), then the algorithm can be used as reserve. We also notice that the algorithm doesn't make mistakes in the planning, and the time needed to create the schedule using the algorithm is solidly found to be around one hour. Whether the VCST management accepts or rejects the algorithm and the solutions proposed will depend on the accuracy of the schedule that is currently manually constructed.

APPENDICES

APPENDIX A MACHINES PER WORK STATION IN THE FFG

Work station number.	Operation	Machine number – Machine at VCST
1	Soft turning	1 – MURATEC 20001 2 – MURATEC 20002 3 – MURATEC 20003 4 – MURATEC 20004 5 – MURATEC 20005
2	Milling	6 – PFAUTER 20006 7 – PFAUTER 20007 8 – PFAUTER 20008 9 – PFAUTER 20009
3	Scraping	10 – SICMAT 20010 11 – SICMAT 20011 12 – SICMAT 20012
4	Broaching	13 – FORST 20029 14 – FORST 20030
5	Drilling	15 – STAMA 20039 16 – STAMA 20040 17 – STAMA 20081
6	STC	18 – Surface Treatment Company
7	Hard turning	19 – WEISSER 20037 20 – WEISSER 20098 21 – 6374 MURATEC MS10
8	Pressing	22 – Schmidt press 23 – Voco press
9	Grinding	24 – Reishauer RZ362(1) 25 – Reishauer RZ362(2) 26 – Reishauer RZ150 27 – Voumard
10	Inspection 100%	28 – Manual inspection

11	Lasering	29 30 31 32 – TRUMPF 20260	– – –	TRUMPF TRUMPF TRUMPF	20250 20257 20259
12	Extern FFG	33 – External operations			

Table 18. Overview of the machines per work station

APPENDIX B RESULTS FOR BRANDIMARTE MK01

Max. makespan	Avg. makespan	Objective value	A	B	C	D	E	F	G	H	Bench-mark counter	recover point	# iterations
72	51,9	51	1								15	20 it	1000
102	69,1	56	1								15	10 it	1000
94	54,3	71	1								15	10 it	1000
73	50,4	11	1								15	20 it	1000
120	52,8	697	1	1							15	20 it	1000
90	58,1	286,34	1	1 ; pow 0,75							15	30 it	500
73	54,4	249,62	1	1 ; pow 0,75							15	30 it	500
161	108,8	404,58	1		0,5 ; pow 0,25						15	30 it	500
62	52,7	153,69	1			1; pow 0,5					5	30 it	500
64	48,2	120,275				1; pow 0,5					5	30 it	1000
74	56,1	247,972	1		0,5; pow 0,25	1; pow 0,5					5	30 it	500
122	72,7	728					1				5	30 it	750
109	57,7	309	0,5				0,5				5	30 it	750
56	44,7	117,32	0,85			1; pow 0,5	0,15				5	30 it	750
92	55,4	252,493	0,95			1; pow 0,5	0,25				5	30 it	500
62	43,8	162,916	0,75			1; pow 0,5	0,15				5	50 it	500
51	40,3	73,9556	0,65			1; pow 0,5	0,1				5	45 it	500
56	36,6	83,594	0,5			1; pow 0,5	0,08				5	45 it	750
48	36,9	63,9689	0,5			1; pow 0,5	0,08				5	45 it	1000
48	35,3	42,6838	0,4			1; pow	0,06				5	45 it	750

						0,5							
46	39,8	112,773	0,5			1; pow 0,5	0,08		1		5	45 it	750
52	42,6	83,37	0,5			1; pow 0,5	0,08		0,5		5	45 it	750
47	31,7	114,907	0,8			2; pow 0,5	0,12		1		5	45 it	750
44	31,8	106,921	0,8			2; pow 0,5	0,12		1		5	45 it	750
46	37,3	141,455	0,8			2; pow 0,5	0,12		1		5	45 it	1000
52	47,4	176,163	0,8			2; pow 0,5	0,12	2; pow 0,5			5	45 it	750
48	35,3	171,208	0,8			2; pow 0,5	0,12		2		5	45 it	1000
44	35,7	47,2802	0,5			1; pow 0,5	0,08				5	45 it	600
46	36,4	90,2991	0,5			1; pow 0,5	0,08			1	5	45 it	750
49	41,1	79,934	0,5			1; pow 0,5	0,08	1; pow 0,5			5	45 it	750

Table 19. Results for test instance Brandimarte mk01

APPENDIX C EXTRACT FROM INPUT DATA FOR THE VCST CASE

nbJobs 34
nbWSs 12
nbMachines 33

OEE for machines:
70 70 70 70 70 75 75 75 75 80 80 80 85 85 85 85 85 100 80 80 85 95 95 60 60 75 75 100 95 95 95 95 100

setup for machines:
2.9 2.9 2.9 2.9 2.9 2.75 2.75 2.75 2.75 2.75 2.75 2.75 1.3 1.3 2.35 2.35 2.35 0 2.75 2.75 2.75 0 0 3 3 3 3 0 0 0 0 0 0

q trial for machines:
0.75 0.75 0.75 0.75 0.75 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.85 0.85 0.5 0.5 0.5 0 0.5 0.5 0.5 0 0 1 1 1 1 0 0 0 0 0 0

machines per work station
1 2 3 4 5
6 7 8 9
10 11 12
13 14
15 16 17
18
19 20 21
22 23
24 25 26 27
28
29 30 31 32
33

composition of jobs and operations

job 299417	→ ***job id***
7	→ ***number of operations in this job***
900	→ ***lot size***
1	→ ***priority weight of the job***
1000	→ ***release date (in minutes)***
200000	→ ***due date (in minutes)***
6-85.0 7-85.0 8-85.0 9-85.0	→ ***set of operations of the job, each line = 1 operation:***
10-70.0 11-70.0 12-70.0	***(qualified) machine number – duration of operation***
13-98.0 14-98.0	***if performed on this machine***
15-151.0 16-151.0	
18-7200.0	
29-57.8 30-57.8 31-57.8 **32-57.8**	→ ***The 6th operation's duration is 57.8 minutes, if performed on***
33-7200.0	***machine 32***

job 299648
8 ….

APPENDIX D RESULTS FOR THE VCST CASE, OEE = 100%

Makespan (*)	*Gap*	*coeff$_D$*	*coeff$_G$*
852	1,59	2,25	2,5
707,57	1,32	2,25	2,42
686,07	1,28	2,25	**2,37**
805,16	1,50	2,25	2,3
867,02	1,62	2,25	2,25
719,01	1,34	2,25	2,12
770,93	1,44	2,25	2,05
690,86	1,29	2,25	**2**
712,15	1,33	2,25	1,95
796,61	1,48	2,25	1,87
759,31	1,41	2,25	1,75
817,82	1,52	2,25	1,62
781,43	1,46	2,25	1,5
764,61	1,42	2,25	1,37
755,85	1,41	2,25	1,25
763,28	1,42	2,25	1,12
823,47	1,53	2,25	1
727,98	1,36	2,25	0,85
812,27	1,51	2,25	*0,80*
760,77	1,42	2,25	0,80
692,25	1,29	2,25	**0,75**
806,92	1,50	2,25	0,7
853,67	1,59	2,25	*0,7*
750,65	1,40	2,25	0,62
756,49	1,41	2,25	0,5
768,47	1,43	2,25	0,37
752,7	1,40	2,25	0,25
823,97	1,54	2,25	0,12

Table 20. VCST results – OEE = 100%, coeff$_D$ = 2.25

Makespan (*)	*Gap*	*coeff$_D$*	*coeff$_G$*
749,7	1,40	4,5	0,75
798,28	1,49	4,25	0,75
753,42	1,40	4	0,75
723,65	1,35	3,75	0,75
810,32	1,51	3,62	0,75
831,86	1,55	3,56	0,75
674,93	1,26	**3,5**	**0,75**
732,49	1,36	3,43	0,75
748,75	1,40	3,37	0,75
833,38	1,55	3,25	0,75
679,09	1,27	3	0,75
822,31	1,53	2,75	0,75
732,67	1,37	2,5	0,75
690,89	1,29	2,25	0,75
687,72	1,28	2	0,75
776,52	1,45	1,75	0,75
735,21	1,37	1,5	0,75
742,76	1,38	1,25	0,75
741,13	1,38	1	0,75
864,81	1,61	0,75	0,75
821,01	1,53	0,5	0,75
884,3	1,65	0,25	0,75

Table 21. VCST results – OEE = 100%, coeff$_G$ = 0.75

(*) *coeff$_A$=0.8, coeff$_E$* =0.1, secondary parameters as defined in section 4.1.3 with 750 iterations per simulation and back to recover point after 50 iterations.

Makespan (*)	*Gap*	*coeff$_D$*	*coeff$_G$*
809,43	1,51	4,5	2
739,16	1,38	4,25	2
735,99	1,37	4	2
837,27	1,56	3,92	2
678,05	1,26	**3,87**	**2**
811,41	1,51	3,82	2
699,69	1,30	3,75	2
784,64	1,46	3,7	2
738,87	1,38	3,62	2
823,91	1,54	3,5	2
814,7	1,52	3,25	2
710,87	1,32	3	2
732,23	1,36	2,75	2
733,69	1,37	2,5	2
784,38	1,46	2,37	2
900,81	1,68	2,32	2
688,3	1,28	2,25	2
744,68	1,39	2,19	2
702,76	1,31	2,12	2
768,91	1,43	2	2
942,34	*1,76*	*2*	*2*
725,7	1,35	1,75	2
729,19	1,36	1,50	2
774,19	1,44	1,25	2
798,32	1,49	1	2
728,42	1,36	0,75	2
792,31	1,48	0,5	2
783,06	1,46	0,25	2

Table 22. VCST results – OEE = 100%, coeff$_G$ = 2

Makespan (*)	*Gap*	*coeff$_D$*	*coeff$_G$*
851,65	1,59	4,5	2,37
788,82	1,47	4,25	2,37
905,94	1,69	4,12	2,37
700,4	1,30	4	2,37
727,39	1,36	3,87	2,37
780,02	1,45	3,75	2,37
857,03	1,60	3,5	2,37
841,31	1,57	3,25	2,37
851,24	1,59	3	2,37
807,13	1,50	2,75	2,37
824,73	1,54	2,5	2,37
703,56	1,31	2,25	2,37
795,64	1,48	2	2,37
751,62	1,40	1,75	2,37
690,32	1,29	1,62	2,37
678,26	1,26	1,5	2,37
683,08	1,27	1,37	2,37
700,88	1,31	1,25	2,37
729,69	1,36	1	2,37
844,95	1,57	0,75	2,37
795,45	1,48	0,5	2,37
808,37	1,51	0,37	2,37
879,53	1,64	0,32	2,37
592,91	1,10	**0,25**	**2,37**
834,56	1,55	0,19	2,37
727,84	1,36	0,12	2,37

Table 23. VCST results – OEE = 100%, coeff$_G$ = 2.37

Makespan (*)	*Gap*	*coeff$_D$*	*coeff$_G$*
688,54	1,28	**3,5**	**0,75**
684,22	1,27	3,5	0,75
695,71	1,30	3,5	0,75
724,8	1,35	3,5	0,75
804,47	1,50	3,5	0,75
850,23	1,58	3,5	0,75
785,87	1,46	3,5	0,75
721,46	1,34	3,5	0,75
790,49	1,47	3,5	0,75
726,57	1,35	3,5	0,75

Table 24. VCST results – OEE = 100%, coeff$_D$=3.5, coeff$_G$ = 0.75

Makespan (*)	*Gap*	*coeff$_D$*	*coeff$_G$*
722,79	1,35	**3,87**	**2**
978,42	1,82	3,87	2
805,62	1,50	3,87	2
817,36	1,52	3,87	2
778,65	1,45	3,87	2
979,15	1,82	3,87	2
789,16	1,47	3,87	2
769,36	1,43	3,87	2
945,6	1,76	3,87	2
803,92	1,50	3,87	2

Table 25. VCST results – OEE = 100%, coeff$_D$=3.87, coeff$_G$ = 2

Makespan (*)	*Gap*	*coeff$_D$*	*coeff$_G$*
837,81	1,56	**0.25**	**2,37**
710,73	1,32	0,25	2,37
824,22	1,54	0,25	2,37
845,55	1,58	0,25	2,37
796,87	1,48	0,25	2,37
757,05	1,41	0,25	2,37
767,52	1,43	0,25	2,37
865,03	1,61	0,25	2,37
810,96	1,51	0,25	2,37
847,95	1,58	0,25	2,37

Table 26. VCST results – OEE = 100%, coeff$_D$=0.25, coeff$_G$ = 2.37

APPENDIX E RESULTS FOR THE VCST CASE, OEE < 100%

Makespan (*)	*Gap*(+)	$coeff_D$	$coeff_G$
880,798	1,64	2,25	2,5
855,307	1,59	2,25	2,37
1012,47	1,89	2,25	2,25
925,319	1,72	2,25	2,12
915,504	1,71	2,25	2
1014,75	1,89	2,25	1,87
788,049	1,47	2,25	**1,75**
883,387	1,65	2,25	1,62
835,194	1,56	2,25	1,5
865,963	1,61	2,25	1,5
993,558	1,85	2,25	1,37
885,632	1,65	2,25	1,25
871,009	1,62	2,25	1,15
1020,54	*1,90*	2,25	*1*
818,98	1,53	2,25	1
806,798	1,50	2,25	**0,85**
813,343	1,52	2,25	0,75
858,49	1,60	2,25	0,62
868,854	1,62	2,25	0,5
876,301	1,63	2,25	0,25
839,178	1,56	2,25	0,12
901,537	1,68	2,25	0

Table 27. VCST results – OEE < 100%, $coeff_D$ = 2.25 (default)

(+) Since no exact solution is known for the case where OEE < 100%, the gap is expressed with respect to the exact solution of 536,72h for OEE = 100%

Makespan (*)	*Gap*(+)	$coeff_D$	$coeff_G$
877,13	1,63	5	0,85
825,977	1,54	4,75	0,85
956,812	1,78	4,5	0,85
1181,24	2,20	*4,25*	0,85
859,31	1,60	4,25	0,85
871,009	1,62	4,12	0,85
768,094	1,43	4,06	0,85
756,809	1,41	4	0,85
836,703	1,56	3,94	0,85
933,36	1,74	3,87	0,85
836,37	1,56	3,75	0,85
991,358	1,85	*3,75*	0,85
893,401	1,66	3,62	0,85
756,805	1,41	**3,5**	**0,85**
812,383	1,51	3,37	0,85
759,036	1,41	3,25	0,85
810,513	1,51	3	0,85
838,64	1,56	2,75	0,85
891,277	1,66	2,5	0,85
977,243	1,82	2,25	0,85
1008,7	1,88	2	0,85
1022,23	1,90	1,75	0,85
1039,98	1,94	1,5	0,85
876,398	1,63	1,25	0,85
942,141	1,76	1	0,85
840,68	1,57	0,75	0,85
1052,52	1,96	0,5	0,85
824,564	1,54	0,25	0,85
806,252	1,50	0,2	0,85
1046,44	1,95	0,1	0,85
999,138	1,86	0	0,85
991,358	1,63	5	0,85
893,401	1,54	4,75	0,85
756,805	1,78	4,5	0,85

Table 28. VCST results – OEE < 100%, $coeff_G$ = 0.85

Makespan (*)	*Gap*(+)	$coeff_D$	$coeff_G$
905,99	1,69	4,5	1,75
816,705	1,52	4,25	1,75
824,324	1,54	4	1,75
950,876	1,77	3,75	1,75
1167,89	2,18	*3,75*	*1,75*
1095,25	2,04	3,5	1,75
901,461	1,68	3,25	1,75
1046,3	1,95	3	1,75
960,866	1,79	2,75	1,75
819,575	1,53	2,62	1,75
779,949	1,45	2,5	1,75
874,406	1,63	2,37	1,75
829,354	1,55	2,25	1,75
808,472	1,51	2	1,75
824,182	1,54	1,75	1,75
999,674	1,86	1,5	1,75
1080,49	2,01	1,25	1,75
976,495	1,82	1	1,75
1073,65	2,00	0,75	1,75
937,872	1,75	0,5	1,75
848,346	1,58	0,25	1,75

Table 29. VCST results – OEE < 100%, $coeff_G$ = 1.75

Makespan (*)	*Gap*(+)	$coeff_D$	$coeff_G$
909,547	1,69	3,5	0,85
875,089	1,63	3,5	0,85
881,006	1,64	3,5	0,85
807,504	1,50	3,5	0,85
917,807	1,71	3,5	0,85
941,554	1,75	3,5	0,85
936,632	1,75	3,5	0,85
770,479	1,44	3,5	0,85
871,009	1,62	3,5	0,85
918,361	1,71	3,5	0,85

Table 30. VCST results – OEE < 100%, $coeff_D$=3.5, $coeff_G$ = 0.85

Makespan (*)	*Gap*(+)	$coeff_D$	$coeff_G$
893,599	1,69	2,5	1,75
945,387	1,52	2,5	1,75
1117,95	1,54	2,5	1,75
822,493	1,77	2,5	1,75
947,409	2,18	2,5	1,75
986,418	2,04	2,5	1,75
1002,97	1,68	2,5	1,75
990,5	1,95	2,5	1,75
996,247	1,79	2,5	1,75
944,785	1,53	2,5	1,75

Table 31. VCST results – OEE < 100%, coeffD=2.5, coeffG = 1.75

BIBLIOGRAPHY

[1] http://www.madeinlimburg.be/nieuws/vcst-investeert-30-miljoen-in-roemeense-fabriek/ , 27/03/2013

[2] http://www.madeinlimburg.be/nieuws/vcst-start-met-productie-in-china/, 27/03/2013

[3] Bart Vandenbroucke, Onderzoek naar toepasbaarheid van assembly line balancing modellen in customized laag volumeproductie, 2011

[4] Goldratt, E.M., Critical chain, North River Press, 1997

[5] Policella, N. Scheduling with Uncertainty A Proactive Approach using Partial Order Schedules. Phd thesis, Università degli Studi di Roma "La Sapienza", 2005

[6] Flexibility and robustness in scheduling, J. Billaut, A. Moukrim & E. Sanlaville, 2008]

[7] J.C. Beck, N. Wilson. Proactive algorithms for job shop scheduling with probabilistic durations, Journal of Artificial Intelligence Research 28(1), 183–232, 2007

[8] M. R. Garey, D. S. Johnson and Ravi Sethi. The Complexity of Flowshop and Jobshop Scheduling, Mathematics of Operations Research May 1976 vol. 1 no. 2 117-129

[9] http://en.wikipedia.org/wiki/NP-complete, 4/4/2013

[10] A Generic Multi-Agent Reinforcement Learning Approcach for Scheduling Problems, Yailen Martínez Jiménez, 2012

[11] Brucker, P., Scheduling algorithms, Springer (2004).

[12] Optimization of Job Shop Scheduling in Focused Flexile Gearshop, Thomas Verlodt

[13] Reeves, C.R., Modern heuristic techniques for combinatorial problems, Halsted Press, 1993

[14] Johnson S. M. Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly 1954;1:61 – 8

[15] Palmer D.S.; Sequencing jobs through a multi-stage process in the minimum total time – a quick method of obtaining a near optimum. Operations Research Quarterly 1965;16(1):101-7

[16] Gupta JND. A functional heuristic algorithm for the flowshop scheduling problem. Operational Research Quarterly 1971;22(1):39–47.]

[17] H. G. Campbell, R.A. Dudek, and M. L. Smith; A heuristic algorithm for the n job, m machine sequencing problem. Management science Vol 16. No 10, June 1970

[18] Dannenbring DG. An evaluation of flow shop sequencing heuristics. Management Science 1977;23(11):1174–82

[19] Nawaz M, Enscore EEJ, Ham I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. OMEGA, The International Journal of Management Science 1983;11(1):91–5]

[20] Suliman SMA. A two-phase heuristic approach to the permutation flow-shop scheduling problem. International Journal of Production Economics 2000;64:143–52].

[21] David, S.J., et al. Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning, Operations Research, **37**(6) (1989), p. 865-892

[22] Mastrolilli, M. and L.M. Gambardella, Effective neighbourhood functions for the flexible job shop problem, Journal of Scheduling, **3**(1) (2000), p. 3-20.

[23] Saidi-Mehrabad, M. and P. Fattahi, Flexible job shop scheduling with tabu search algorithms, International Journal of Advanced Manufacturing Technology, **32**(5), 2007, p. 563-570.

[24] Glover, F. and M. Laguna, Tabu Search, Kluwer Academic Publishers, 1998

[25] Luger, G.F., Artificial Intelligence: Structures and Strategies for Complex Problem Solving, Addison-Wesley (2005).

[26] Cawsey, A., The essence of artificial intelligence, Prentice Hall (1998).

[27] Rubén Ruiz, Concepción Maroto, Javier Alcaraz. Two new robust genetic algorithms for the flowshop scheduling problem, Omega 34 (2006) 461 – 476

[28] A. Sofia Simaria and P. M. Virarinho. A genetic algorithm based approach to the mixed-model assembly line balancing problem of type II, Computers & Ind. Engineering 47, 2004, 391-407

[29] An efficient multiobjective genetic algorithm for mixed-model assembly line balancing problem considering demand ratio-based cycle time, Wenqiang Zhang and Mitsuo Gen,Springer Science+ Business Media, LLC 2009

[30] Murata T, Ishibuchi H, Tanaka H. Genetic algorithms for flowshop scheduling problems. Computers & Industrial Engineering 1996;30(4):1061–71

[31] Reeves CR. A genetic algorithm for flow shop sequencing. Computers & Operations Research 1995;22(1):5–13.

[32] Barto, Sutton, & Watkins, 1990; Grefenstette, Ramsey, & Schultz, 1990

[33] Russell, S. & Norvig, P., Artificial Intelligence: A Modern Approach, Prentice Hall, 2003

[34] Y. Shoham, R. Powers, T. Grenager. Multi-Agent Reinforcement Learning: a critical survey, 2003

[35] Jennings, Sycara & Wooldridge. A roadmap of agent research and development. Autonomous Agents and Multi-agent Systems, , 17–38, 1998

[36] (An introduction to) Reinforcement Learning, R. Sutton, A. Barto, The MIT Press, Cambridge, Massachusetts, 1998

[37] D.E. Moriarty, A.C. Schultz, J.J. Grefenstette. Evolutionary Algorithms for Reinforcement Learning, Journal of Artificial Intelligence Research 11 (1999), pg 241-276

[38] http://en.wikipedia.org/wiki/Overall_equipment_effectiveness, 17/4/2013

[39] D. Behnke, M. J. Geiger. Test Instances for the Flexible Job Shop Scheduling Problem with Work Centers, Research Report, 2012

[40] D. Kahneman, Thinking, Fast and Slow; 14th edition, Business Contact, Amsterdam 2011, pg. 233-238

LIST OF FIGURES

LIST OF TABLES