FACTLOG
www.factlog.eu

ENERGY-AWARE FACTORY ANALYTICS FOR PROCESS INDUSTRIES

Deliverable D5.1

Real-time re-optimization algorithms

Version Version 1.0	**Lead Partner** NISSA
Date 30/06/2021	**Project Name** FACTLOG – Energy-aware Factory Analytics for Process Industries

FACTLOG has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement Nº 869951

Call Identifier H2020-NMBP-SPIRE-2019	**Topic** DT-SPIRE-06-2019 - Digital technologies for improved performance in cognitive production plants
Project Reference 869951	**Start date** November 1st, 2019
Type of Action IA – Innovation Action	**Duration** 42 Months

Dissemination Level

X	**PU**	Public
	CO	Confidential, restricted under conditions set out in the Grant Agreement
	CI	Classified, information as referred in the Commission Decision 2001/844/EC

Disclaimer

Executive Summary

This deliverable reports on the specification and implementation of the State-of-the-art Optimization Methods of the FACTLOG project, reflecting the work performed in the context of the project Task 5.1 Robust Optimization Methods, and the outcomes thereof. Following the analysis of the pilot scenarios relevant to the needs for optimization, in parallel with the elaboration of the different use cases and system requirements, the progressive implementation of the different modules which are dependent on optimization as well as the ones which optimization depends on, this deliverable proceeds with describing the optimization methods as well as the initial version of the optimization toolkit of the FACTLOG ecosystem.

Starting from the toolkit itself, it is designed to be modular, expandable, and capable to be adapted and introduced in different cases, starting from the pilots themselves. It can solve short- mid- and long- term production optimization problems and depending on the different pilots (ongoing and post the projects lifecycle) and their needs, it can address the provision of optimal production schedules (e.g., BRC) to re-scheduling (e.g., PIA) and re-configuration of setting of different production units (e.g., TUPRAS), taking into account all process and business constraints. The Optimization as a Service module, named optEngine consists of different layers (as presented in D1.3 Architecture and Technical Specification) is responsible for the interconnection of the Optimization Module to the remaining FACTLOG modules (e.g ECTs) enabling them to initiate an optimization round or receive the optimization's round results.

Besides the development of the optimization engine this Task had the goal of providing a theoretical approach to solving the different identified optimization problems of the pilots. Starting from the TUPRAS case relevant to the Oil Refineries and to the LPG Production process, the role of optimization was to handle the recovery to on-specs LPG production in the most energy efficient way. To do so, the optimization module identifies the most energy efficient combination of operational scenarios for all process units involved in the LPG purification process and proposes the settings combinations to the shopfloor (or ECT) to take the appropriate decision and action. The TUPRAS case was solved utilizing a MIP model, based on a typical flow, and blending modelling approach that also incorporates a binary decision variable for each operational scenario of each process unit. This approach enables the optimization engine to take under consideration all involved units in the process, something that also pushed the boundaries of research (as current solutions focus on single unit optimization). Through experimentation we found that the behavior of our proposed approach given different time horizons for recovery will be suitable for real field application.

Moving forward from TUPRAS to the second pilot and respective problem solved by the optimization module we have the PIA case. In the PIACENZA case, a main goal as in the overall modern textile industry is to increase productivity while reducing production costs. The case has been challenging in terms of optimization as it has the inherent properties of weaving scheduling (job splitting and sequence dependent setup times) in parallel to additional setup constrains as in this case the number of setups that can be performed simultaneously on different machines is restricted due to a limited number of setup workers and daily setup time is also bounded. The PIACENZA case was solved utilizing a mixed integer linear programming (MILP) formulation that captures the elaborate structure of the weaving process extended by two combinatorial heuristics that differ on the way they perform job splitting and assignment to machines, to handle large real instances. Through

experimentation we found that the solutions provide the best policies to balance makespan, number of tardy jobs and total tardiness over weekly instances.

The third case in the FACTLOG project the Optimization Toolkit deals with is the CONTINENTAL case. This is a discrete automotive part manufacturing environment that is modelled as a 2-stage assembly flow shop with resource constraints. Key challenge is the integration of maintenance planning and scheduling together with the scheduling of production orders at the production lines. To that end, an analytics module is providing maintenance windows and the goal is to schedule maintenance activities during periods that will have a minimum impact on the schedule in terms of makespan and tardiness. Another capability that is provided is dynamic re-scheduling to capture new urgent orders and unscheduled machine breakdowns. A rigorous Constraint Programming formulation is proposed for modeling and solving the problem. Preliminary results on benchmark data sets validate the applicability of the model and demonstrate the efficiency, effectiveness and scalability of the proposed CP approach.

Lastly, the fourth case in the FACTLOG project the Optimization Toolkit handles is the BRC Steel production case. This is a multistage flowshop with parallel machines at each stage. The main challenge in this particular case was the lack of digitized information. That combined with the inherent difficulty in needing cranes to unload / load machines create important bottlenecks in the production process. The goal of optimization in this case was to find the optimal production schedule in relation to the makespan or the number of tardy jobs. The BRC case was solved utilizing MILP for an extended flexible multistage flowshop problem with machine dependent setup times. Preliminary experimentation showed that the MIP model can handle instances of medium size quite easily and can provide production policies that balance between the criterion of minimum makespan and tardy jobs.

In addition to the optimization toolkit, we provide a novel approach for detecting variations in the process structure (phases) based on processing energy consumption data. These variations define the structure of a process and as such are important for the optimization of the process execution, as defined in our Cognitive Factory Model (reported in deliverable D3.1). Sensors for measuring energy consumption are very common in the industry processes. Since it is planned to install additional energy sensors in BRC and PIACENZA pilots, these two will be used for the validation phase.

That is, all sections related to optimisation document the research and innovation work on optimization (AUEB, UNIPI), while the last section brings in an indicative enhancement and interplay of the analytics, as presented in D3.1 (NISSA).

Revision History

Revision	Date	Description	Organisation
0.1	1/06/2021	ToC	NISSA
0.2	10/06/2021	Provision of Optimization Input for the cases of PIA, TUPRAS, BRC and CONT Chapters 1-6	AUEB, UNIPI
0.3	20/06/2021	Provision of Analytics input Chapter 7	NISSA
0.4	25/06/2021	Released for Internal review	AUEB
0.5	27/06/2021	Peer review	JSI
1.0	28/06/2021	Addressing review comments and finalized document for submission	NISSA

Contributors

Organisation	Author	E-Mail
UNIPI	Pavlos Eirinakis	pavlose@unipi.gr
UNIPI	Grigoris Koronakos	gregkoron@gmail.com
UNIPI	Konstantinos Kaparis	k.kaparis@uom.edu.gr
UNIPI	Penny Kalpodimou	pennykalp@unipi.gr
AUEB	Gregory Kasapidis	gkasapidis@aueb.gr
AUEB	Panagiotis Repousis	prepousi@aueb.gr
AUEB	Yiannis Mourtos	mourtos@aueb.gr
AUEB	Stavros Lounis	slounis@aueb.gr
AUEB	Georgios Zois	georzois@aueb.gr
NISSA	Nenad Stojanovic	nenad.stojanovic@nissatech.com

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose and Scope

WP5 will provide the optimization for the FACTLOG project. To this end, T5.1 Real-time re-optimization as part of cognitive twins constitutes the first approach in solving the different cases based on the problems identified taking under consideration the overall FACTLOG approach (and other modules). In this task, the optimization related to the extension of the abilities of the Enhanced Cognitive Twins is conducted and the output is the actual solving approaches taking under consideration textbook and state-of-the-art solution approaches in terms of exact methods and heuristics.

This deliverable reports on the former aspects reflecting the work performed in T5.1, Real-time re-optimization as part of cognitive twins following the insight gained by the previous deliverables and respective work done in interconnected Tasks and Work packages. D5.1 elaborates on the optimization solutions capable to be invokes by the ECTs or Human Operators under different circumstances.

1.2 Relation with other Deliverables

The starting point for this deliverable is the ‘D1.1 Reference Scenarios, KPIs and Datasets’, where the pilot cases were examined from the needs of optimization as well and initial KPIs as well as available datasets were identified to feed optimization. Furthermore, ‘D1.2 Cognitive Factory Framework ‘presents the role of optimization in the FACTLOG cognition cycle as well as its potential interactions with other modules. Additionally, D5.1 strongly relates with ‘D1.3 FACTLOG System Architecture and Technical Specifications’ where the Optimization modules interconnection with all other modules is initially presented. Lastly it relates with the ‘D6.5 Integrated Package and Platform’ where the actual interconnection of the Optimization Toolkit in the FACTLOG is explained. D5.1 provides input to ‘D5.2 Robust and energy-aware planning and scheduling’ and ‘D5.3 FACTLOG optimization toolkit and service’. Also, and in parallel to other deliverables, it will feed the optimization approach in respective deliverables as ‘D3.4 Proactive Cognitive Plants’, ‘D6.2 Data collection Framework’ and ‘D6.6 Integrated Package and Platform’.

1.3 Structure of the Document

Section 2 presents the optEngine, being the connection interface of the Optimization Toolkit to the FACTLOG remaining modules. From then on, the sections document the work done on each pilot in terms of optimization. Specifically, they present (a) the Pilot case from the optimization perspective, (b) the identified optimization problem, (c) the literature review conducted by the optimization team to account for a thorough scientific coverage of currently existing solutions, (d) the solution approach for the case, (e) the mathematical formulation of the pilot problem, and (f) its solution and benchmarking. This presentation flow appears in Section 3 regarding the TUPRAS optimization, in Section 4 on the PIACENZA case, in Section 5 concerning the CONTINENTAL case and in Section 6 presenting the BRC case. Extending the optimization related aspects, Section 7 present an enhancement and an interplay with indicative analytics of FACTLOG. It concludes with the Appendices, where for each case the structure of the input and output of the optEngine for all cases is presented.

2 Optimization-As-a-Service

In this section we describe the optimization module developed and deployed in the framework of this project. Hereafter, we will refer to this module as ***optEngine***.

OptEngine works as a shell around the optimization services build for the purposes of this project. Its architecture follows an asynchronous approach and is agnostic to optimization-specific data requirements. That is, optEngine receives, stores and forwards the optimization data to the optimization service requested by the end-user. The structure of this section goes as follows: in **Section 2.1** we provide a detailed description of optEngine's architecture; in **Section 2.2** we provide optEngine's technology stack; in **Section 2.3** we provide a detailed documentation of optEngine's web API with which the end-user interacts.

2.1 Functional Requirements

The use cases of this shell are listed below:

- Use Case 1 (UC1): Authenticate user.
- Use Case 2 (UC2): List the available optimization services.
- Use Case 3 (UC3): Submit a new optimization job.
- Use Case 4 (UC4): Get the status and progress of an optimization job.
- Use Case 5 (UC5): Cancel an ongoing optimization job.
- Use Case 6 (UC6): Get the solution of a complete optimization job.
- Use Case 7 (UC7): Store the solution of a complete optimization job.

The image below illustrates the set of these functional requirements:

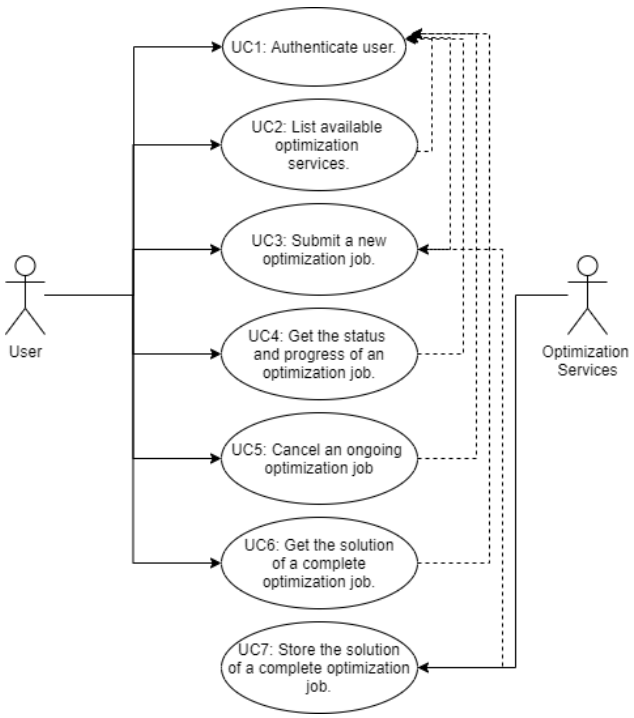

Figure 1. optEng Functional Requirements

Use Case 1	Authenticate user.	
Brief description	This use case states the actions taken to authenticate a user.	
Primary Actors	User	
Pre-conditions	The user is registered to optEngine.	
Post-conditions	The user is authenticated.	
Basic flows	Tasks	Information required
	1. User includes the base64-encoded username:password combination to every action.	Username, password
	2. The system authenticates the user.	
Alternative flows	Tasks	Information required
	1. If an error occurs, the system returns the respective error code.	Error_code

Use Case 2	List the available optimization services.	
Brief description	This use case states the actions taken in order to list the available optimization services.	
Primary Actors	User	
Pre-conditions	The user is registered and authenticated.	
Post-conditions	The user receives a list with the available optimization services.	
Basic flows	Tasks	Information required
	1. User requests the list with the available optimization services.	
	2. The system returns the available optimization services	Route_ids
Alternative flows	Tasks	Information required
	If an error occurs, the system returns the respective error code.	Error_code

Use Case 3	Submit a new optimization job.
Brief description	This use case states the actions taken to submit a new optimization job.
Primary Actors	User
Pre-conditions	The user is registered and authenticated.
Post-conditions	The user receives a unique identifier, the status, and the progress of the submitted optimization job.

Basic flows	Tasks	Information required
	1. User submits the optimization data along with the optimization service route id.	Optimization_data, route_id
	2. The system forwards the submitted optimization job to the respective optimization service.	Optimization_data, route_id
	3. The system returns the unique identifier, the status and progress of the submitted job.	Uuid, status, progress
Alternative flows	Tasks	Information required
	If an error occurs, the system returns the respective error code.	Error_code

Use Case 4	Get the status and progress of an optimization job.	
Brief description	This use case states the actions taken to retrieve, the status and progress of a submitted optimization job.	
Primary Actors	User	
Pre-conditions	The user is registered and authenticated. The user has submitted an optimization job.	
Post-conditions	The user receives unique identifier and the status of the submitted optimization job.	
Basic flows	Tasks	Information required
	1. User submits the unique identifier of the optimization job.	uuid
	2. The system returns the unique identifier, the status and progress of the submitted job.	Uuid, status, progress
Alternative flows	Tasks	Information required
	If an error occurs, the system returns the respective error code.	Error_code

Use Case 5	Cancel an ongoing optimization job.
Brief description	This use case states the actions taken to cancel an ongoing optimization job.
Primary Actors	User
Pre-conditions	The user is registered and authenticated. The user has submitted an optimization job. The optimization job is not finished yet.
Post-conditions	The user receives unique identifier and the status of the submitted optimization job.

Basic flows	Tasks	Information required
	1. User submits the unique identifier of the optimization job.	uuid
	2. The system returns the unique identifier, the status and progress of the submitted job.	Uuid, status, progress
Alternative flows	Tasks	Information required
	If an error occurs, the system returns the respective error code.	Error_code

Use Case 6	Get the solution of a complete optimization job.	
Brief description	This use case states the actions taken to get the solution of a submitted optimization job	
Primary Actors	User	
Pre-conditions	The user is registered and authenticated. The user has submitted an optimization job. The optimization job is successfully complete.	
Post-conditions	The user receives unique identifier and the solution data.	
Basic flows	Tasks	Information required
	1. User submits the unique identifier of the optimization job.	uuid
	2. The system returns the unique identifier and the solution data of the submitted job.	Uuid, solution_data
Alternative flows	Tasks	Information required
	If an error occurs, the system returns the respective error code.	Error_code

Use Case 7	Store the solution of a complete optimization job.	
Brief description	This use case states the actions taken to store the solution of a complete optimization job	
Primary Actors	Optimization Services	
Pre-conditions	The user has submitted an optimization job. The optimization job is successfully complete.	
Post-conditions	The solution data is successfully stored.	
Basic flows	Tasks	Information required

	1. The Optimization Services submit the uuid and the solution data.	Uuid, solution_data
	2. The system stores the solution data and updates the status of the optimization job to complete.	Uuid, solution_data, status
Alternative flows	Tasks	Information required
	If an error occurs, the system returns the respective error code.	Error_code

The class diagram below depicts the data requirements of optEngine.

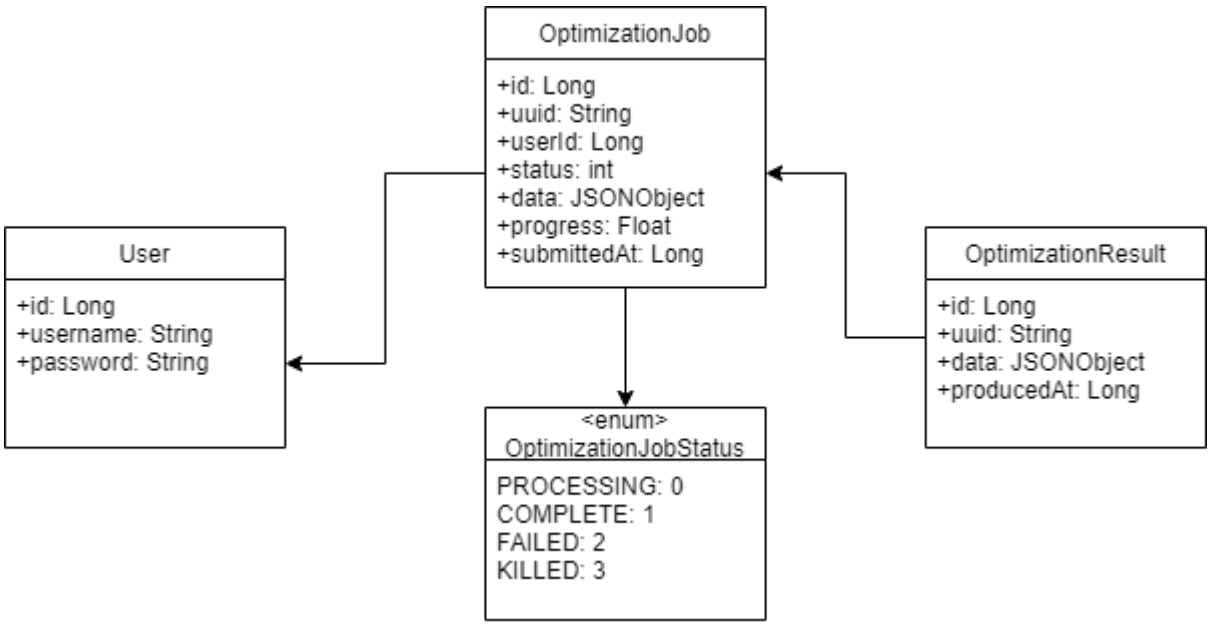

Figure 2. optEngine Data Requirements

2.2 Architectural view

OptEngine works as a shell around the optimization. Its architecture follows an asynchronous approach and is agnostic to optimization-specific data requirements. That is, optEngine receives, stores, and forwards the optimization data to the optimization service requested by the end-user.

Optimization requests along with the respective data are received via a web API. This API allows the actions described in the previous section. The communication with the API requires authentication, is encrypted (https) and asynchronous, i.e., once an optimization job is submitted, the callee does not wait for its completion.

Data stores within optEngine work in a twofold manner:

- Permanent storage via a database (db): this is where optimization requests and the related data are permanently stored or retrieved and updated when necessary.

- Temporal storage via the use of queues: this is where optimization data is stored up to the point where they get consumed by the optimization services that read these queues.

Regarding the optimization data, both the db and the queues are data-agnostic following a general json schema. This allows the storage, permanent and temporal, of different data structures required from different optimization services.

The employed queues allow the asynchronous processing of an optimization job. Additionally, by being durable they ensure that when optEngine or an optimization service fails, the job along with data are available in the respective queue. This means that optEngine, upon reception of a new optimization job, forwards it to the requested optimization service via a queue. Each optimization service listens for a new optimization job to a specific queue and writes status/progress updates to another queue. Last, optEngine listens to (a) a queue for status/progress updates and (b) multiple queues for optimization results.

The flow of data and the architectural approach of optEngine are depicted below.

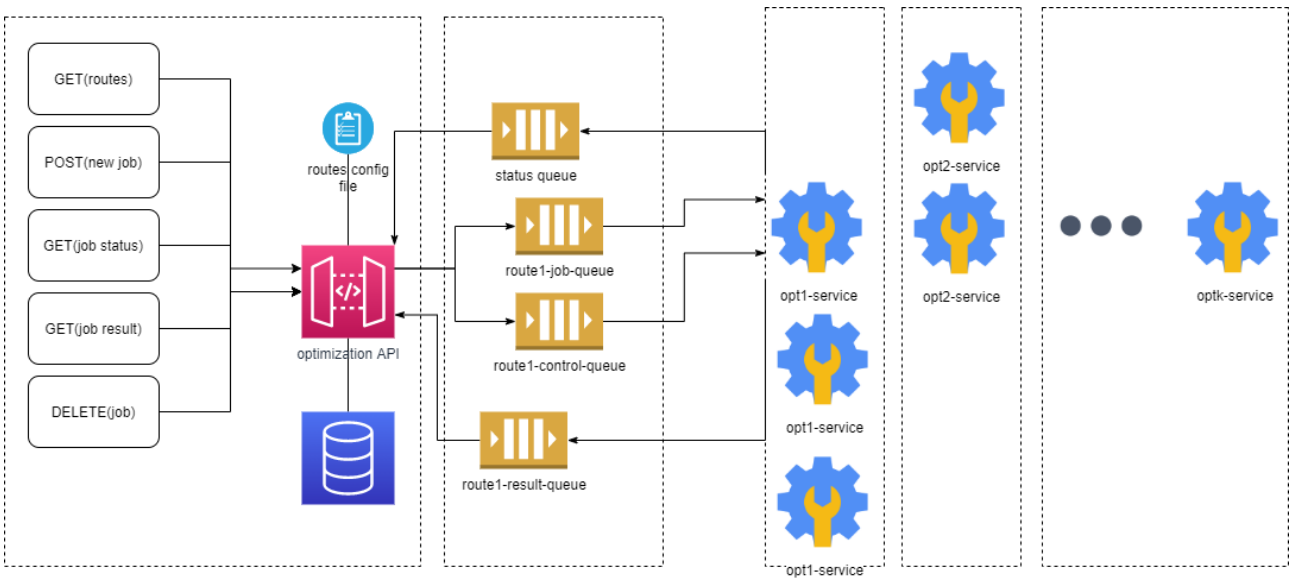

Figure 3. optEngine data Flow

2.3 Technology Stack

The technologies used to develop optEngine are the following:

- Java 8
- Spring-boot 2.4.5
- Springdoc openAPI 1.5.2
- Hibernate 1.0.0
- PostgreSQL 11
- RabbitMQ 3.8.16
- Docker 18.09.7

Figure 4. optEngine data stack

2.4 Web API documentation

2.4.1 API calls documentation

Method	GET		
Path	/route/list		
Description	Get the routes list.		
Parameters	Name	Description	
	Authorization*(header)*	The base-64 encoded string with the user credentials username:password used for Basic authorization.	
Responses	Status	Body	Description
	200	RoutesDTO	Found the result of the optimization job with the supplied uuid.
	500	AdoptApiError	An internal error has occurred.

Method	POST		
Path	/opt/job		
Description	Submit a new optimization job.		
Parameters	Name	Description	
	Authorization*(header)*	The base-64 encoded string with the user credentials username:password used for Basic authorization.	
Body	JobSubmissionDTO		
Responses	Status	Body	Description
	200	JobStatusDTO	Successfully submitted new job.
	400	AdoptApiError	Invalid route or no data supplied.
	500	AdoptApiError	An internal error has occurred.

Method	GET		
Path	/opt/job		
Description	Get the status of a submitted optimization job.		
Parameters	Name	Description	
	Uuid	The unique identifier of the optimization job.	
	Authorization*(header)*	The base-64 encoded string with the user credentials username:password used for Basic authorization.	
Responses	Status	Body	Description
	200	JobStatusDTO	Found the optimization job with the supplied uuid.
	400	AdoptApiError	Invalid/no optimization job uuid supplied.
	404	AdoptApiError	No optimization job with the supplied uuid found.
	500	AdoptApiError	An internal error has occurred.

Method	DELETE		
Path	/opt/job		
Description	Kill a submitted optimization job.		
Parameters	Name	Description	
	Uuid	The unique identifier of the optimization job.	
	Authorization*(header)*	The base-64 encoded string with the user credentials username:password used for Basic authorization.	
Responses	Status	Body	Description
	200	JobStatusDTO	Optimization job with supplied uuid successfully killed.
	400	AdoptApiError	Invalid/no optimization job uuid supplied.
	404	AdoptApiError	No optimization job with the supplied uuid found.
	500	AdoptApiError	An internal error has occurred.

Method	GET
Path	/opt/job/result
Description	Get the result of a completed optimization job.

Parameters	Name	Description	
	Uuid	The unique identifier of the optimization job.	
	Authorization*(header)*	The base-64 encoded string with the user credentials username:password used for Basic authorization.	
Responses	Status	Body	Description
	200	JobResultDTO	Found the result of the optimization job with the supplied uuid.
	400	AdoptApiError	Invalid/no optimization job uuid supplied.
	404	AdoptApiError	No result found for the specified optimization job uuid.
	500	AdoptApiError	An internal error has occurred.

2.4.2 JSON bodies description

Name	RoutesDTO	
Description	Contains information about the available routes.	
Attributes	Name	Description
	uuid	Unique identifier of the call.
	routes	The array with the available routes.

Example	{
	"routes": [
	"string"
	],
	"uuid": "c11dc261-d8a4-4174-897f-3864defee150"
	}	

Name	JobSubmissionDTO	
Description	Contains information about the optimization job and is submitted to trigger the optimization service.	
Attributes	Name	Description
	route	Unique identifier of the optimization job category.
	data	The required data for the optimization job. The schema is custom to each type of optimization job.

Example	{
	"route": "max-flow",
	"data": {
	"empty": true,
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	}
	}

Name	JobStatusDTO	
Description	Contains information related to the status of the optimization job.	
Attributes	Name	Description
	submitted_at	The submission date of the optimization job in millis.
	progress	The percentage (%) of optimization job progress.
	uuid	Unique identifier of the optimization job.
	status	The status of the optimization job. 0=PROCESSING, 1=COMPLETE, 2=FAILED, 3=KILLED.

Example	{
	"submitted_at": 1623427969000,
	"progress": 67.5,
	"uuid": "c11dc261-d8a4-4174-897f-3864defee150",
	"status": 0
	}	

Name	JobResultDTO	
Description	Contains information about the solution of the optimization job.	
Attributes	Name	Description
	uuid	Unique identifier of the optimization job.
	produced_at	The production date of the optimization result in millis.
	data	The required data for the optimization job. The schema is custom to each type of optimization job.

Example	{
	"data": {
	"empty": true,
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	},
	"produced_at": 1623427969000,
	"uuid": "c11dc261-d8a4-4174-897f-3864defee150"
	}	

<table>
<tr><td>Name</td><td colspan="2">AdoptApiError</td></tr>
<tr><td>Description</td><td colspan="2">Contains information about API errors.</td></tr>
<tr><td rowspan="5">Attributes</td><td>Name</td><td>Description</td></tr>
<tr><td>path</td><td>The URL.</td></tr>
<tr><td>message</td><td>The error message.</td></tr>
<tr><td>uuid</td><td>The unique identifier of the optimization job.</td></tr>
<tr><td>status</td><td>The http status.</td></tr>
<tr><td>Example</td><td colspan="2"><pre>{
  "path": "/opt/job",
  "message": "Internal Server Error",
  "uuid": "c11dc261-d8a4-4174-897f-3864defee150",
  "status": 500
}</pre></td></tr>
</table>

3 Oil Refineries: Pilot Case by TUPRAS

3.1 Introduction

The Liquified Petroleum Gas (LPG) is a fuel produced as a by-product of natural gas and oil refining industry. The production of LPG must adhere to certain quality specifications regarding specific impurities (e.g., Sulphur, Naphtha and Ethane). In this regard, LPG purification is a complex process involving different types of interconnected process units. Within this process, anomalies that may arise in any given process unit may lead to off-specs situations, i.e., the LPG in the collection tank may deviate from the desired specifications. In such cases, actions need to be taken to drive on-specs recovery, thus ensuring the final mixture complies with the regulations.

In the context of the FACTLOG project, the TUPRAS pilot outlines the critical aspects which should be targeted in order to recover from an off-specs situation in the most energy-efficient manner. These aspects relate to (a) the early detection of off-specs LPG production, (b) the exploration of the root-cause and (c) the on-specs recovery of LPG production. Hence, within FACTLOG, the role of Optimization is to handle point (c). That is, the digital twin of the LPG production process is enhanced with optimization capabilities to drive on-specs recovery while minimizing the energy consumption required for this effort. In a nutshell, our optimization approach is based upon the identification of the most energy efficient combination of operational scenarios for all process units involved in the LPG purification process to drive on-specs recovery of the produced LPG.

More specifically, in the TUPRAS refinery that is examined within FACTLOG, the LPG accumulated in the final tank is obtained from the aggregation of outputs of ten different input feeds. Each input feed is purified via a set of subsequent and (in some cases) interconnected process units that remove impurities (i.e., debutanizers, deethanizers and LPG DEA units). Each process unit can be considered (and hence modelled) as a function that transforms input to output; this function can either be stated explicitly by a mathematical formulation or implicitly derived by data-driven approaches such as a machine learning (ML) model. In this regard, the operational settings of each different process unit (e.g., top and/or bottom temperature, reboiler flow, pressure etc) constitute the inputs of each process unit, while the resulting reduction of impurities in the flow of the LPG stream, the reduction of LPG flow itself, and the energy consumed constitute the output of each process unit. The mapping of all possible inputs to their corresponding outputs for a specific process unit constitute the unit's operational scenarios. That is, each operational scenario is fully specified by (i) the operational settings of the process unit (pressure, temperatures, etc.), (ii) the reduction in flow rates and impurities of outputs from the unit (products) and (iii) the corresponding energy (heat and/or electricity) consumption.

Figure 5 presents these three basic ingredients of each operational scenario. These scenarios are produced by the modelling module that may incorporate both mathematical functions as well as analytical and ML models for each process unit. However, it must be noted that Optimization only requires the reduction in flow and impurities and the corresponding energy consumption for each scenario. This information for each operational scenario of each process unit along with real-time production data (e.g., flow and composition of input feeds, quantity of LPG and impurities in the final tank etc) and real-time data (e.g., energy consumed) are the inputs of our optimization solution method. Based on this input and the production process scheme, our optimization method produces and solves

a Mixed Integer Programming (MIP) model (detailed in Section 3.3.3). The obtained solution has a global perspective, since optimization examines the whole process and provides the operational scenario that needs to be adopted for each process unit involved in LPG production to drive on-specs recovery in the most energy efficient manner. In particular, the solution constitutes of an assignment of a specific operational scenario to each process unit involved. Each such operational scenario corresponds to specific operational settings (e.g., increase of temperature at the top of the unit by a specified number of degrees) which need to be applied to the corresponding unit; these proposed settings can also be used by the Simulation module to evaluate the proposed solution.

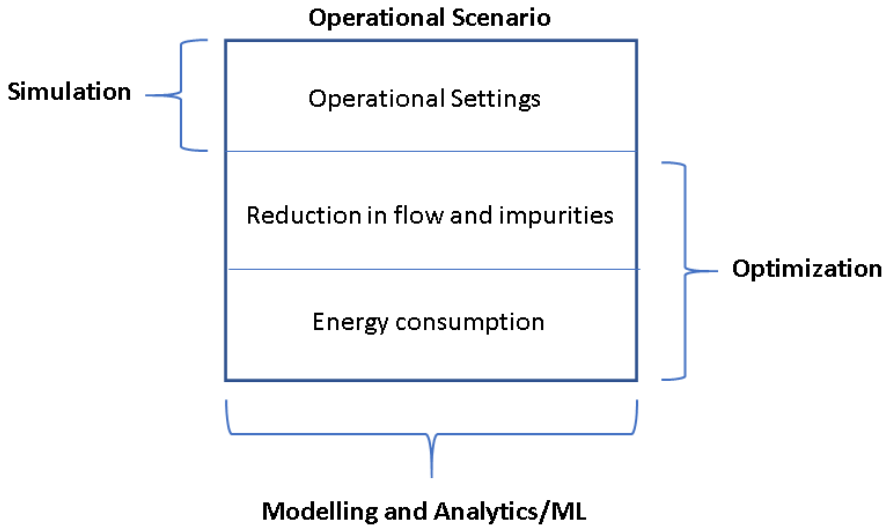

Figure 5. Operational scenarios and the modules producing them (modelling and analytics/ML) and consuming them (optimization and simulation)

3.2 Literature Review

Several challenges of oil refining industry have been met by employing optimization. Khor et al (2011) and Albahri et al (2018) determine the optimal topology configuration of petroleum refineries based on mixed integer linear programming (MILP). Concerning the energy management of refinery operations, Iyer and Grossmann (1997) and Mete and Turkay (2018) employ MILP to derive the optimum combinations of the equipment that minimize the energy costs. Sales et al (2018) employed nonlinear optimization and simulation of refinery units to obtain a production planning for a refinery that maximizes profit. Also, planning and scheduling issues concerning the petroleum supply network of typical refineries are addressed by utilizing MILP in Moro and Pinto (2004) as well as in Kuo and Chang (2008). Li et al (2005) criticised the use of linear models for crude distillation unit (CDU), Fluid Catalytic Cracking (FCC) and product blending in refinery planning. They proposed instead simplified empirical nonlinear process models. Kemaloglu et al (2009) designed a controller for predictive control of crude oil preheat and distillation column of a crude oil unit in Tupras Izmit Refinery.

In the context of FACTLOG, we focus on the LPG production process. Pinto and Moro (2000) developed a MILP model to generate a schedule for LPG refinery management so as to optimize the selection of storage facilities that are used to receive these products and to feed the product pipeline. In addition, several studies are devoted to the maximization of the production of LPG specifically in the FCC unit. Such studies are conducted by Gouvea

and Odloak (1998), Zanin et al (2000) and Zanin et al (2002), and they are based on non-linear optimization models to accommodate the operation of the corresponding predictive controller. In a similar vein, Vasconcelos et al (2005) resort to sequential quadratic programming for the maximization of the LPG and gasoline profit. Almeida Neto et al (2000) studied a debutanizer unit, from the gasoline stream producing LPG, by incorporating linear models to Model Predictive Control (MPC).

3.3 Optimization model and solution method

3.3.1 LPG purification process

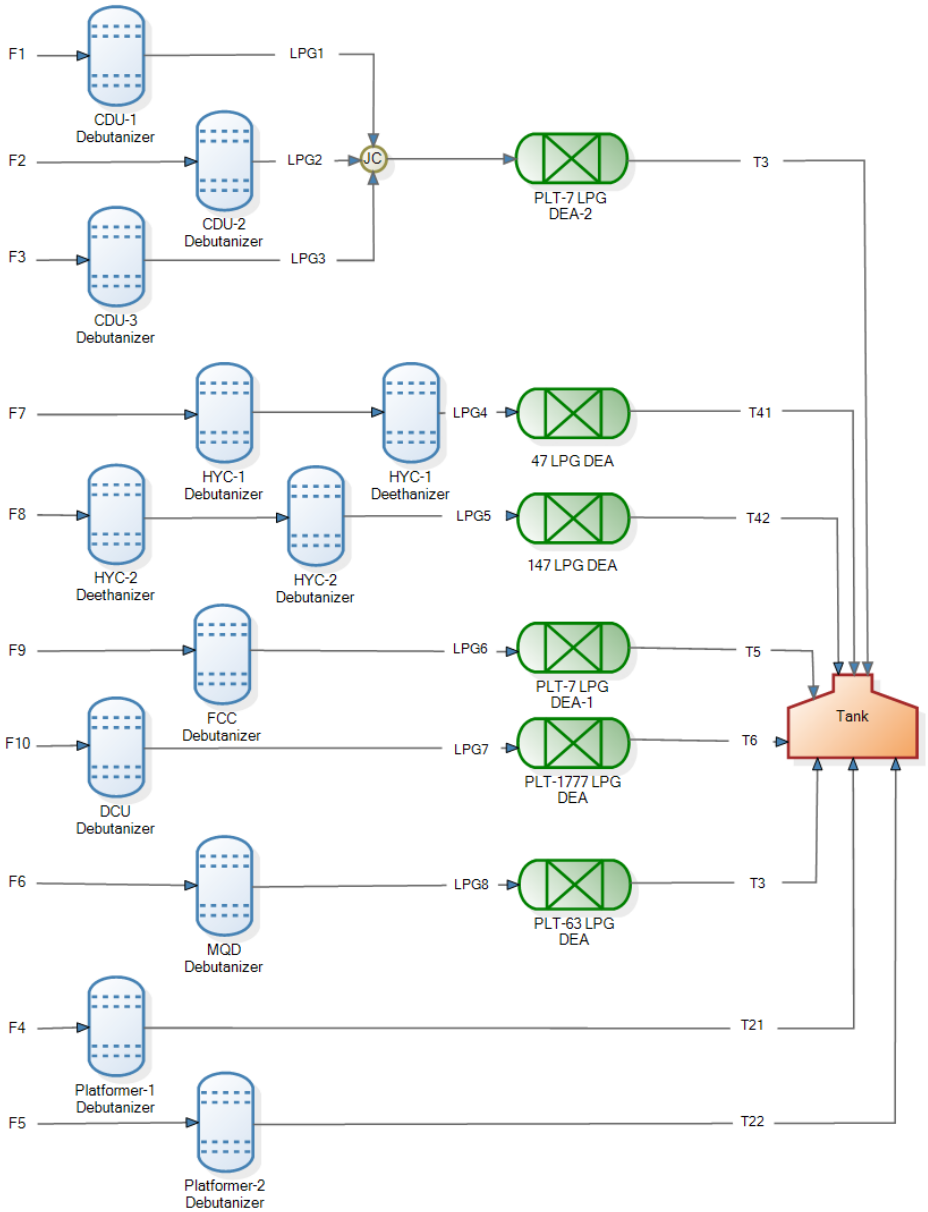

Figure 6. Level 2 process model of TUPRAS LPG purification plant

The LPG purification process is a complex procedure that involves different interconnected process units, i.e., debutanizers, deethanizers and LPG DEA. Figure 6 provides a schematic of this process. It includes three CDU debutanizer units and two platformer debutanizer units

whose outputs are mixed, respectively. Moreover, it includes an FCC, a DCU and an MQD debutanizer. There are also two hydrocracker (HYC) debutanizers and deethanizers (these process the input feed sequentially in two different streams). Moreover, each of the ten input streams ranging from F1 to F10 represents the individual stream that enters each specific unit. Junctions of streams, denoted by *JC* symbol, represent LPG streams (e.g., LPG1, LPG2 and LPG3) that are mixed before entering the corresponding LPG DEA unit (responsible for removing sulphur from the LPG stream).

Hence, initially, each feed is treated to remove carbon-based impurities (e.g., C2, C5, etc.). Depending on the input feed, this step may require processing from a debutanizer (see F1-F6, F9, F10 in Figure 6) or from both a debutanizer and a deethanizer (see F7-F8 in Figure 6). Next, depending on the type of the debutanizer in the first process stage, the output of the LPG may be further processed by a LPG DEA unit to remove sulphur-based impurities. At the final stage of the process, the purified LPG flows are aggregated in the final LPG tank.

3.3.2 Optimization Model

Contrary to the existing work in the literature, our modelling approach is not confined to a specific process unit but incorporates information from all units that are involved in LPG production process, offering a global optimization tool. Once an off-specs situation has been identified, the optimization module is triggered to provide a plan on how to recover to on-specs LPG production at a global level, with the minimum energy requirements within a given time-frame. In fact, the proposed plan is derived from the optimal solution and can be mapped to explicit operational settings for each process unit (e.g., configurations for each process unit about temperature, pressure, etc.). We consider each specific set of operational settings of each unit as an operational scenario. For instance, a debutanizer receives a specific feed and applies temperature (at the top and at the bottom) as well as pressure to remove impurities, i.e., C2 (and lighter by-products) from the top and C5 (and heavier by-products) from the bottom. The different settings that may be applied (e.g., higher/lower temperature at the top/bottom with different levels of pressure applied) result in different outcomes (level of impurities removed) and accordingly correspond to different energy consumption levels.

In general, higher energy consumption (i.e., higher costs) leads to a higher removal of impurities, hence on-specs recovery demands more energy (cost). However, the optimal level of energy and the required actions for recovery are unknown. For this purpose, we build our model with the aim to minimize the energy consumption while satisfying the production specifications of LPG and by incorporating the operational scenarios of each process unit. We represent the LPG purification process via a MIP model, based on a typical flow and blending modelling approach that also incorporates a binary decision variable for each operational scenario of each process unit. In this manner, for each unit, the optimization module selects whether a specific operational scenario is applied or not. Thus, the proposed model determines collectively the optimal combination of settings for all process units by directly selecting the optimal operational scenarios for each one of them.

Note that this modelling approach has enabled us to avoid directly incorporating temperature, pressure etc and their relation to energy consumption as variables in the model, and hence to avoid introducing within the model the non-linear relationships that these impose. That is, we have removed the non-linearity from the model (hence obtaining a simpler model to optimize) in a rather intuitive manner, by introducing binary decision

variables (based on the operational scenarios) that model the actual decisions that a process engineer needs to take when planning for on-specs recovery, i.e., how to change the operational settings of each process unit.

In what follows, we provide the notation of the sets, the constants and the variables that are employed in the MIP model. The quantities are measured in kilos (kg), the time intervals are given in hours and flow rate in kilos per hour (kg/hour). Also, the feed to the debutanizers is assumed stable for the whole period.

The sets that are used to formally define the MIP model are provided below:

Sets:
I_{proc}: Set of process units (includes debutanizers, deethanizers, LPG DEA, etc.)
I_{junc}: Set of junctions that aggregate different flows into one
$I_{units} = I_{proc} \cup I_{junc}$: Set of all process units and junctions
I_{input}: Set of input units of the process (i.e., the raw feed flows)
I_{output}: Set of output units of the process (i.e., in this case, only the final LPG tank)
S_i: Set of operational scenarios for unit i, $\forall\, i \in I_{units}$
N_i^- : Set of units (neighbors) to which unit $i \in I_{input} \cup I_{units}$ sends flow
N_i^+ : Set of units (neighbors) from which unit $i \in I_{units} \cup I_{output}$ receives flow
$Po(N_i)$: Set of units (neighbors), $N_i \equiv N_i^-$ or $N_i \equiv N_i^+$, that are in the path of the process towards any output unit in I_{output} to which $i \in I_{input} \cup I_{units}$ sends flow

The constants that are used to formally define the MIP model are provided below:

Constants:
$H \in \mathbf{N}^+$: Time horizon (in hours)
$E_i^s \in \boldsymbol{R}$: Energy consumption of unit I_{units} running operational scenario $s \in S_i$ for a unit of time (i.e., for an hour)
$IF_i \in \boldsymbol{R}$, $i \in I_{input}$: Input flow rate (kg/hour) of raw feed $i \in I_{input}$
$CAP_{ij} \in \boldsymbol{R}$, $i \in I_{units}$: Capacity, i.e., maximum flow rate (kg/hour) that unit i can forward towards unit $j \in N_i^-$
$PF_{ij}^s \in \boldsymbol{R}$, $i,j \in I_{units}$: Percentage of flow rate that flows through unit i under operational scenario s towards unit $j \in N_i^-$
$Q_{start}^i \in \boldsymbol{R}$: Quantity (kg) of LPG in $i \in I_{output}$ (i.e., the LPG tank) at the start of the recovery
$Q_{total}^i \in \boldsymbol{R}$: Total quantity (kg) that $i \in I_{output}$ (i.e., the LPG tank) can hold
$ISU_i \in \boldsymbol{R}$, $i \in I_{input}$: Input flow rate (kg/hour) of Sulphur within the LPG flowing from the raw feed $i \in I_{input}$
$PSU_{ij}^s \in \boldsymbol{R}$: Percentage of flow rate of Sulphur for unit i running operational scenario $s \in S_i$ towards unit $j \in N_i^-$. Accordingly, the percentage reduction of Sulphur is $1 - PSU_{ij}^s$. Note that PSU_{ij}^s = 100% if the process unit does not remove Sulphur (e.g., for debutanizers).
$QSU_{start}^i \in \boldsymbol{R}$: Quantity (kg) of Sulphur in $i \in I_{output}$ (i.e., the LPG tank) at the start of the recovery
$SP_{SU} \in \boldsymbol{R}$: Specifications (%) for max percentage of Sulphur in the LPG tank
$IC2_i \in \boldsymbol{R}$, $i \in I_{input}$: Input flow rate (kg/hour) of C2 within the LPG flowing from the raw feed $i \in I_{input}$

$\boldsymbol{PC2_{ij}^{s}} \in \boldsymbol{R}$, $\boldsymbol{i} \in \boldsymbol{I_{units}}$: Percentage of flow rate of C2 for unit *i* running operational scenario $s \in S_i$ towards unit $j \in N_i^-$. Accordingly, the percentage reduction of C2 is $1 - PC2_{ij}^{s}$. Note that $PC2_{ij}^{s} = 100\%$ if the process unit does not remove C2, e.g., for LPG DEA

$\boldsymbol{QC2_{start}^{i}} \in \boldsymbol{R}$: Quantity (kg) of C2, in $i \in I_{input}$ (i.e., the LPG tank) at the start of the recovery

$\boldsymbol{SP_{C2}} \in \boldsymbol{R}$: Specifications (%) for max percentage of C2 in the LPG tank

$\boldsymbol{IC5_i} \in \boldsymbol{R}$, $\boldsymbol{i} \in \boldsymbol{I_{input}}$: Input flow rate (kg/hour) of C5 within the LPG flowing from the raw feed $i \in I_{input}$

$\boldsymbol{PC5_{ij}^{s}} \in \boldsymbol{R}$, $\boldsymbol{i} \in \boldsymbol{I_{units}}$: Percentage of flow rate of C5 for unit *i* running operational scenario $s \in S_i$ towards unit $j \in N_i^-$. Accordingly, the percentage reduction of C5 is $1 - PC5_{ij}^{s}$. Note that $PC5_{ij}^{s} = 100\%$ if the process unit does not remove C5, e.g., for LPG DEA

$\boldsymbol{QC5_{start}^{i}} \in \boldsymbol{R}$: Quantity (kg) of C5, in $i \in I_{input}$ (i.e., the LPG tank) at the start of the recovery

$\boldsymbol{SP_{C5}} \in \boldsymbol{R}$: Specifications (%) for max percentage of C5 in the LPG tank

$\boldsymbol{SP_{C2+C5}} \in \boldsymbol{R}$: Specifications (%) for max percentage of sum of C2 and C5 in the LPG tank

The variables that are used to formally define the MIP model are provided below:

Variables:

$\boldsymbol{x_i^s} \in \{\boldsymbol{0}, \boldsymbol{1}\}$: Unit *i* ∈ unit I_{units} runs operational scenario $s \in S_i$ or not

$\boldsymbol{q_i} \in \boldsymbol{R}$: Quantity of LPG (kg) in output $i \in I_{output}$ (i.e., final LPG tank) at the end of the recovery (i.e., at the end of hour H)

$\boldsymbol{qSU_i} \in \boldsymbol{R}$: Quantity of Sulphur in output $i \in I_{output}$ (i.e., final LPG tank) at the end of the recovery (i.e., at the end of hour H)

$\boldsymbol{qC2_i} \in \boldsymbol{R}$: Quantity of C2 in output $i \in I_{output}$ (i.e., final LPG tank) at the end of the recovery (i.e., at the end of hour H)

$\boldsymbol{qC5_i} \in \boldsymbol{R}$: Quantity of C5 in output $i \in I_{output}$ (i.e., final LPG tank) at the end of the recovery (i.e., at the end of hour H)

$\boldsymbol{f_{ij}^{s}} \in \boldsymbol{R}$: Flow rate (kg/hour) of unit $i \in I_{units}$ running operational scenario $s \in S_i$ towards unit $j \in N_i^-$

$\boldsymbol{f_{ij}^{*}} \in \boldsymbol{R}$: Flow rate (kg/hour) of unit $i \in I_{input} \cup I_{units}$ towards unit $j \in N_i^-$

$\boldsymbol{fSU_{ij}^{s}} \in \boldsymbol{R}$: Flow rate (kg/hour) of Sulphur flowing through unit $i \in I_{input} \cup I_{units}$ running operational scenario $s \in S_i$ towards unit $j \in N_i^-$

$\boldsymbol{fSU_{ij}^{*}} \in \boldsymbol{R}$: Flow rate (kg/hour) of Sulphur flowing through unit $i \in I_{input} \cup I_{units}$ running operational towards unit $j \in N_i^-$

$\boldsymbol{fC2_{ij}^{s}} \in \boldsymbol{R}$: Flow rate (kg/hour) of C2 flowing through unit $i \in I_{input} \cup I_{units}$ running operational scenario $s \in S_i$ towards unit $j \in N_i^-$

$\boldsymbol{fC2_{ij}^{*}} \in \boldsymbol{R}$: Flow rate (kg/hour) of C2 flowing through unit $i \in I_{input} \cup I_{units}$ towards unit $j \in N_i^-$

$\boldsymbol{fC5_{ij}^{s}} \in \boldsymbol{R}$: Flow rate (kg/hour) of C5 flowing through unit $i \in I_{input} \cup I_{units}$ running operational scenario $s \in S_i$ towards unit $j \in N_i^-$

$\boldsymbol{fC5_{ij}^{*}} \in \boldsymbol{R}$: Flow rate (kg/hour) of C5 flowing through unit $i \in I_{input} \cup I_{units}$ towards unit $j \in N_i^-$

We present below the MIP model that enables identifying the optimal combination of operational scenarios for the process units that are involved in the LPG purification process.

$$\min H \cdot \sum_{i \in I_{units}} \sum_{s \in S_i} E_i^s \cdot x_i^s$$

s.t.

$$\sum_{s \in S_i} x_i^s = 1 \qquad \forall i \in I_{units} \qquad (3.1)$$

$$f_{ij}^* = IF_i \qquad \forall i \in I_{input}, j \in N_i^- \qquad (3.2)$$

$$f_{ij}^* = \sum_{s \in S_i} f_{ij}^s \qquad \forall i \in I_{units}, j \in N_i^- \qquad (3.3)$$

$$f_{ij}^s \leq CAP_{ij} \cdot x_i^s \qquad \forall i \in I_{units}, j \in N_i^- \qquad (3.4)$$

$$\sum_{k \in N_i^+} f_{ki}^* = \sum_{s \in S_i} \frac{1}{PF_{ij}^s} \cdot f_{ij}^s \qquad \forall i \in I_{units}, j \in N_i^- \qquad (3.5)$$

$$q_i = Q_{start}^i + H \cdot \sum_{j \in N_i^+} f_{ki}^* \qquad \forall i \in I_{output} \qquad (3.6)$$

$$q_i \leq Q_{total}^i \qquad \forall i \in I_{output} \qquad (3.7)$$

$$fSU_{ij}^* = ISU_i, \qquad \forall i \in I_{input}, j \in N_i^- \qquad (3.8)$$

$$fSU_{ij}^* = \sum_{s \in S_i} fSU_{ij}^s, \qquad \forall i \in I_{units}, j \in N_i^- \qquad (3.9)$$

$$fSU_{ij}^* \leq f_{ij}^s, \qquad \forall i \in I_{units}, j \in N_i^- \qquad (3.10)$$

$$\sum_{k \in N_i^+} fSU_{ki}^* = \sum_{j \in N_i^-} \frac{1}{PSU_{ij}^s} \cdot fSU_{ij}^s, \qquad \forall i \in I_{units}, j \in P_o(N_i^-) \qquad (3.11)$$

$$qSU_i = QSU^i_{start} + H \cdot \sum_{j \in N_i^+} fSU^*_{ki} \qquad \forall i \in I_{output} \qquad (3.12)$$

$$qSU_i \leq SP_{SU} \cdot q_i \qquad \forall i \in I_{output} \qquad (3.13)$$

$$fC2^*_{ij} = IC2_i, \qquad \forall i \in I_{input}, j \in N_i^- \qquad (3.14)$$

$$fC2^*_{ij} = \sum_{s \in S_i} fC2^s_{ij}, \qquad \forall i \in I_{units}, j \in N_i^- \qquad (3.15)$$

$$fC2^*_{ij} \leq f^s_{ij}, \qquad \forall i \in I_{units}, j \in N_i^- \qquad (3.16)$$

$$\sum_{k \in N_i^+} fC2^*_{ki} = \sum_{j \in N_i^-} \frac{1}{PC2^s_{ij}} \cdot fC2^s_{ij}, \qquad \forall i \in I_{units}, j \in P_o(N_i^-) \qquad (3.17)$$

$$qC2_i = QC2^i_{start} + H \cdot \sum_{j \in N_i^+} fC2^*_{ki} \qquad \forall i \in I_{output} \qquad (3.18)$$

$$qC2_i \leq SP_{C2} \cdot q_i \qquad \forall i \in I_{output} \qquad (3.19)$$

$$fC5^*_{ij} = IC5_i, \qquad \forall i \in I_{input}, j \in N_i^- \qquad (3.20)$$

$$fC5^*_{ij} = \sum_{s \in S_i} fC5^s_{ij}, \qquad \forall i \in I_{units}, j \in N_i^- \qquad (3.21)$$

$$fC5^*_{ij} \leq f^s_{ij}, \qquad \forall i \in I_{units}, j \in N_i^- \qquad (3.22)$$

$$\sum_{k \in N_i^+} fC5^*_{ki} = \sum_{j \in N_i^-} \frac{1}{PC5^s_{ij}} \cdot fC5^s_{ij}, \qquad \forall i \in I_{units}, j \in P_o(N_i^-) \qquad (3.23)$$

$$qC5_i = QC5^i_{start} + H \cdot \sum_{j \in N_i^+} fC5^*_{ki} \qquad \forall i \in I_{output} \qquad (3.24)$$

$$qC5_i \leq SP_{C5} \cdot q_i \qquad \forall i \in I_{output} \qquad (3.25)$$

$$qC2_i + qC5_i \leq SP_{C2+C5} \cdot q_i \qquad \forall i \in I_{output} \qquad (3.26)$$

$$x^s_i \in \{0, 1\} \qquad \forall i \in I_{units}, s \in S_i \qquad (3.27)$$

$$q_i \geq 0 \qquad \forall i \in I_{output} \qquad (3.28)$$

$$qSU_i \geq 0 \qquad \forall i \in I_{output} \qquad (3.29)$$

$$qC2_i \geq 0 \qquad \forall i \in I_{output} \qquad (3.30)$$

$$qC5_i \geq 0 \qquad \forall i \in I_{output} \qquad (3.31)$$

$$f^*_{ij} \geq 0 \qquad \forall i \in I_{input} \cup I_{units}, j \in N_i^- \qquad (3.32)$$

$$fSU^*_{ij} \geq 0 \qquad \forall i \in I_{input} \cup I_{units}, j \in N_i^- \qquad (3.33)$$

$$fC2^*_{ij} \geq 0 \qquad \forall i \in I_{input} \cup I_{units}, j \in N_i^- \qquad (3.34)$$

$$fC5^*_{ij} \geq 0 \qquad \forall i \in I_{input} \cup I_{units}, j \in N_i^- \qquad (3.35)$$

$$f^s_{ij} \geq 0 \qquad \forall i \in I_{units}, j \in N_i^-, s \in S_i \qquad (3.36)$$

$$fSU^s_{ij} \geq 0 \qquad \forall i \in I_{units}, j \in N_i^-, s \in S_i \qquad (3.37)$$

$$fC2^s_{ij} \geq 0 \qquad \forall i \in I_{units}, j \in N_i^-, s \in S_i \qquad (3.38)$$

$$fC5^s_{ij} \geq 0 \qquad \forall i \in I_{units}, j \in N_i^-, s \in S_i \qquad (3.39)$$

3.3.3 Description of the MIP model

The set of constraints (3.1)-(3.39) define the solution space (i.e., the set of all feasible solutions), while the sum of the energy consumption of all operational scenarios of the process units is minimized in the objective function of the model. Notice that the objective function can be easily modified to support other objectives as well. The set of constraints (3.1) ensure that only one operational scenario will be selected for each process unit.

Constraints (3.2) introduce the input feed. The set of constraints (3.3)-(3.4) ensure that the flow rate f_{ij}^{*} of unit *i* to the unit *j*, will be equal to the flow rate f_{ij}^{S} that corresponds to the unique selected operational scenario for unit *i*. Also, the flow rate that a unit receives from its predecessors is calculated by the set of constraints (3.5), reduced by the reduction of flow at each predecessor. At the end of the recovery, the quantity of LPG that will be contained in the final LPG tank is calculated by the set of constraints (3.6)-(3.7).

Similarly to constraints (3.2)-(3.7), the constraints (3.12)-(3.26) are designed to control the flow rate and the concentration of the impurities in the LPG flow. Specifically, the set of constraints (3.8)-(3.11) guarantee that the flow rate of sulphur for each unit will be decided by the flow rate of the selected operational scenario. Also, the concentration of Sulphur in the final LPG tank, at the end of the recovery, is calculated by the set of constraints (3.12)-(3.13), with constraint (3.13) applying the constraint emanating from the specification for sulphur in LPG. Accordingly, the set of constraints (3.14)-(3.19) handle the flow rate and the concentration of ethane (C2), while the set of constraints (3.20)-(3.25) the corresponding measures of pentane (C5). Finally, constraints (3.26) conform the quantities of ethane and pentane ($qC2_i$ and $qC5_i$) with the permissible limits of these impurities in the LPG.

3.3.4 Reducing the number of operational scenarios

Since the number of processing units is pre-set (according to the LPG production schema of the refinery), the solution space is dependent upon the number of possible operational scenarios per process unit. Reducing the number of operational scenarios per unit will result in reducing the size of the problem and hence offer computational savings and improvement of the performance time of the model.

One way to reduce the number of operational scenarios is by utilizing Data Envelopment Analysis (DEA) assessment (Charnes et al, 1978). In particular, we may consider the operational scenarios as entities to be evaluated. Thus, we may first measure their performance via linear programming models in the context of a DEA assessment. Note that this method has been utilized before in oil refinery production; Han et al (2016) employed DEA for the selection of optimal temperature method of ethylene cracking furnaces. Apart from providing an efficiency score for each scenario, we may also develop a procedure for ranking them (see the methods proposed in (Doyle and Green, 1994; Andersen and Petersen, 1993; Liang et al, 2008). The procedure will serve as a filter for the MIP model by excluding many operational scenarios, i.e., by reducing the number of the decision variables of the model.

Being aware of the real-time demands that may arise for instances with massive number of operational scenarios and especially under a real-time re-optimization setting, we have further explored other possible methods (besides DEA) for discriminating among them without utilizing optimization. Our proposed procedure enables us to reduce considerably the size of the MIP model. In essence, what our procedure does is to identify the dominating operational scenarios that need to be introduced in the model and remove all others.

More specifically, consider the case of a CDU debutanizer, where the operational scenarios include four measures-criteria (f_1-f_4), i.e., energy consumption, reduction rate of C2, reduction rate of C5 and LPG quantity that will be purified. These values are the components of each operational scenario, which we view as a vector in a four-dimensional space. Let OS_1 and OS_2 be two operational scenarios. We say that OS_1 performs better than OS_2 in

criterion f_i (denoted by $f_i(OS_1) > f_i(OS_2)$) if we prefer the value of OS_1 to that of OS_2 for that criterion. Similarly, we may say that OS_1 performs at least as good as OS_2 and denote it by $f_i(OS_1) \geq f_i(OS_2)$. Then, OS_1 dominates OS_2 (denoted by $OS_1 > OS_2$) if for all criteria $\{f_1,\ldots,f_4\}$ it holds that $f_i(OS_1) \geq f_i(OS_2)$, i=1,…,4, with at least one such relationship being satisfied as a strict inequality, i.e., $f_i(OS_1) > f_i(OS_2)$ for some i=1,…,4. Notice that lower levels of energy consumption are desired, thus this criterion is modified accordingly to accommodate the above dominance relationship.

Our proposed method runs in two steps. We first calculate the convex hull of the operational scenarios by employing the *quickhull* algorithm developed by Barber et al (1996) to identify the extreme points, i.e., the extreme operational scenarios in the convex hull that includes all of them. Next, we apply a technique that is based on dominance relationships to derive from the extreme points of the convex hull only the dominant ones. These operational scenarios will be introduced in the MIP model at the last step of the assessment.

3.4 Computational Experience

3.4.1 Experimental setting

In this section, we provide an experimental analysis to examine the scalability of the proposed approach, i.e., both the scalability of the proposed MIP model (Section 3.3.3) and of a pre-processing step that reduces the number of operational scenarios (Section 3.3.4) Moreover, we examine the quality of the calculated solution with respect to the provided time horizon for on-specs recovery.

More specifically, since the number of process units and the corresponding process schema is predefined within the TUPRAS pilot LPG purification process (see Figure 6), we first examine our proposed approach with respect to the number of possible operational scenarios that each process unit may have. In this regard, we have run a set of experiments that examines how the computational time grows with the number of possible operational scenarios per process unit (Section 3.4.2).

Moreover, given the size of the problems that our proposed approach can handle (in terms of the number of operational scenarios per unit) so as to provide the optimal on-specs recovery plan within a realistic solution time frame for real-time optimization, we have developed and examined experimentally a two-step pre-processing method that identifies dominations between the different operational scenarios and removes any operational scenarios that are dominated (Section 3.3.4). These are operational scenarios which the MIP solver would not select for a unit since being dominated by others (e.g., we may remove an operational scenario which removes less impurities and requires more energy than another one for the same unit, since the former should never be chosen over the latter within an optimal solution),

Finally, we examine how the quality of solution changes (regarding the objective function, i.e., the average energy consumed per hour by the whole LPG purification process) with respect to the allowed time horizon to achieve on-specs recovery.

To run the experiments, we have created lab data that we generate randomly by following the directions that TUPRAS has provided. A uniform distribution random number generator has been utilised for all randomisations. In this regard, we have generated random data for:

- all input feeds (LPG flow and impurity composition),
- the operational scenarios of each unit (LPG flow towards tank, impurity composition and energy consumed), and
- the output tank (capacity of the tank, quantity of LPG in the tank at the time that optimization is called, and impurity composition of the tank content).

The code has been developed in C#, with .NET Core 3.1, utilizing the Mixed Integer Programming solver of OR-Tools Version 8.1.84871. Our testing platform is a 3.00 GHz 64-bit Inter(R) Core (TM) i5-8500 machine with 8 GB RAM which runs Windows 10 Pro. In what follows, we report results from 5 independent runs for each case examined.

3.4.2 Evaluating the scalability of the MIP model

We commence our experimental analysis by examining the scalability of the proposed MIP model with respect to the number of operational scenarios for each unit. Table 1 presents the corresponding computational results, providing the time to initialize the MIP model, the time to solve it as well as the total time in seconds. Note that we have run experiments for different time horizons allowed for on-specs recovery (i.e., 6, 12, 18 and 24 hours). Here, we have reported results indicatively for a 12-hours horizon. Note that results did not differ significantly between different time horizons; this was expected, since the time horizon does not affect the size of the problem (i.e., does not alter the number of constraints nor the number of variables).

Our experiments show that for instances in which each unit has at up to 100 operational scenarios, the total required time is approximately 1 second or less. For instances in which each unit has up to 1000 operational scenarios, the time required spans from a few seconds to 1.5 minute, which is still an acceptable time for real-time optimization. This situation changes for a larger number of operational scenarios. Indicatively, for 2000 operational scenarios per unit, the time required is approximately 10.5 minutes, while for 5000 operational scenarios per unit, the solution time is more than 1.5 hour. The latter is not surprising, given the size of the corresponding problem. Indicatively, an instance with 5000 operational scenarios per unit involves solving a MIP with approximately 500.000 variables.

# Operational scenarios per unit	Time to initialize (in seconds)	Time to solve (in seconds)	Total time (in seconds)
10	0.002	0.073	0.075
20	0.004	0.195	0.199
30	0.005	0.174	0.179
40	0.007	0.329	0.336
50	0.008	0.342	0.35
60	0.014	0.632	0.646
70	0.015	0.843	0.858
80	0.016	0.833	0.849
90	0.023	0.747	0.77
100	0.021	1.133	1.154

[1] https://developers.google.com/optimization/

# Operational scenarios per unit	Time to initialize (in seconds)	Time to solve (in seconds)	Total time (in seconds)
200	0.045	4.617	4.662
300	0.065	7.87	7.935
400	0.085	12.067	12.152
500	0.119	17.222	17.341
600	0.15	35.627	35.777
700	0.175	55.21	55.385
800	0.19	51.235	51.425
900	0.229	63.415	63.644
1000	0.254	94.565	94.819
2000	0.498	630.01	630.508
3000	0.703	1503.603	1504.306
4000	0.979	3172.77	3173.749
5000	1.168	5672.515	5673.683

Table 1. Scalability of the MIP model with respect to the number of possible operational scenarios per unit

Since the solution of instances incorporating more than 1000 operational scenarios per unit requires computational time that is non-realistic in a real-time system, in the next section (Section 3.4.3) we examine computationally the procedure proposed in Section 3.3.4 for reducing the number of operational scenarios based on dominance relationships.

3.4.3 Reducing the number of operational scenarios

We apply a two-step procedure that significantly reduces the number of operational scenarios, which should be incorporated in the proposed MIP model for the final assessment. In the first step we calculate the extreme operational scenarios (i.e., convex hull) and in the second step we further discriminate among the scenarios by identifying the dominating ones. Table 2 exhibits the results obtained from experimentation with the operational scenarios of a CDU debutanizer. Each operational scenario includes measures-criteria, i.e., energy consumption, reduction rate of C2, reduction rate of C5 and LPG quantity that will be purified. It is evident that the number of operational scenarios is drastically reduced by our procedure. Thus, a significantly small number of operational scenarios will be introduced in the MIP, rendering it effective for the demands of a real-time optimization system.

# Operational scenarios	# Operational scenarios in convex hull	# Remained operational scenarios	Convex hull time (in seconds)	Dominance time (in seconds)	Total time (in seconds)
3000	280	26	0.010	0.001	0.010
4000	314	30	0.011	0.001	0.012
5000	316	35	0.013	0.001	0.013
10000	380	39	0.014	0.001	0.014
20000	500	44	0.019	0.001	0.020
50000	632	48	0.042	0.001	0.044
75000	704	59	0.070	0.001	0.071
100000	742	58	0.078	0.001	0.080
200000	971	78	0.189	0.002	0.191

# Operational scenarios	# Operational scenarios in convex hull	# Remained operational scenarios	Convex hull time (in seconds)	Dominance time (in seconds)	Total time (in seconds)
500000	1134	83	0.446	0.002	0.448
750000	1364	78	0.577	0.002	0.579
1000000	1342	79	0.890	0.002	0.891
2000000	1577	71	1.692	0.002	1.694
3000000	1756	75	2.285	0.002	2.287
5000000	1869	94	4.410	0.003	4.413

Table 2. Efficiency and time required for the pre-processing steps removing dominated operational scenarios for a CDU debutanizer

3.4.4 Evaluating the quality of solutions vs. the time horizon

In this final part of our experimental analysis, we examine the trade-off between energy consumption and the time required for recovery. We wish to compare the recovery plans produced for different time horizons in terms of the total energy consumption required for recovery (i.e., the objective function which we wish to minimize) over the corresponding hours. That is, we wish to compare the different time horizons based on the hourly energy consumption of the whole LPG purification process, when optimized for on-specs recovery. Note that for the needs of this experiment, and to be able to highlight the difference between alternative recovery plans, we have assumed a very large interval of values for hourly energy consumption per unit (indicatively within the interval [1, 1000]) from which we have randomly selected the consumption corresponding to each operational scenario.

Results are presented in Table 3. It is apparent that as the number of operational scenarios per unit increases, the hourly energy consumption decreases accordingly for all time horizons. This is expected, since more options lead to be better solutions. When examining each case separately, we see that the value of the hourly energy consumption either decreases or remains the same as the time horizon increases. This again is expected. A decrease in value corresponds to the case where, given a larger time horizon, an operational scenario that consumes less energy can be preferred over other scenarios which may be more efficient in removing impurities (and hence drive on-specs recovery faster) but consume more energy. On the other hand, when the value remains the same, the identified operational scenarios are the optimal ones for both time horizons. Overall, the experiments presented in Table 3 validate our expectations on the behavior of our proposed approach given different time horizons for recovery.

	Energy consumption per hours (objective function / hours)			
# Operational scenarios per unit	Horizon 6 h	Horizon 12 h	Horizon 18 h	Horizon 24 h
10	1673	1673	1673	1673
20	688	688	688	554
30	501	501	404	404
40	416	416	416	248
50	291	220	175	175
60	272	272	215	215
70	243	184	142	142
80	273	217	155	155
90	215	215	215	215

	Energy consumption per hours (objective function / hours)			
# Operational scenarios per unit	**Horizon 6 h**	**Horizon 12 h**	**Horizon 18 h**	**Horizon 24 h**
100	193	193	106	70
200	99	99	99	81
300	85	85	85	85
400	58	50	48	52
500	56	56	56	56
600	51	51	37	20
700	45	45	45	37
800	43	43	34	27
900	36	29	29	29
1000	34	34	34	34

Table 3. The effect of the time horizon allowed for on-specs recovery to the objective function

4 Textile Industry: Pilot Case by PIACENZA

4.1 Introduction

Increasing productivity while reducing production costs has been essential in modern textile plants in terms of business sustainability. Scheduling algorithms (Brucker, 1999) offer a viable and effective tool to improve productivity, by optimally allocating the available resources. A typical scheduling problem in textile considers a set of articles/orders to be woven by a set of looms with respect to their delivery dates. Each order is linked to the production of a specific fabric type and is accompanied by a positive quantity, while the looms are unrelated, meaning that each loom operates on different speeds for different orders. The aim is typically to find a schedule with the minimum makespan, i.e., the time that the last executed order is finished.

Two properties make weaving scheduling a challenging problem: job splitting (a job can be split in different machines), and sequence-dependent setup times (per pairs of jobs and per machine). In practice, the latter is justified by the fact that different fabric types require different warp chains for processing, thus imposing machine setup times (to replace the warp chain) from a few hours to a few days (Serafini, 1996). Our work is focused on the weaving scheduling of PIACENZA, a textile enterprise in north Italy that manufactures woollen fabrics for luxury clothing brands. Its production environment is a parallel weaving environment composed of multiple type of looms, operating at different speeds. Weaving scheduling in PIACENZA adopts all the above-described job and machine properties, plus setup resource constraints. Specifically, the number of setups that can be performed simultaneously on different machines is restricted due to a limited number of setup workers and daily setup time is also bounded. We should note that the seminal work of Serafini (1996) signifies the addition of setup resource constraints to the standard weaving scheduling as a severe challenge.

In Section 4.2 we present a brief literature review on weaving scheduling and address the significance and technical novelty of PIACENZA's case. In Section 4.3 we propose a formal definition of our scheduling problem, address its computational complexity and propose a mixed integer linear programming (MILP) formulation that captures the elaborate structure of the weaving process. To handle large real instances, we also propose two combinatorial heuristics that differ on the way they perform job splitting and assignment to machines. We experiment with several weekly instances on both MILP (using a standard solver) and heuristics to establish the computational efficiency of our approach in Section 4.4.

As we note, although typically the trade-off between delivery dates, available machines and setup resources allows the scheduler to deliver each job on time, due to the COVID-19 pandemic many jobs arrive late on the weaving department, while others become tighter in terms of deadline. To improve resource management while avoiding a further increase of tardy jobs, we propose, in Section 4.4, a strategy that dedicates an appropriate number of machines to samples (i.e., jobs with small quantity and tight deadlines) while allocating the rest to regular jobs (i.e., jobs with large quantity and loose deadlines).

4.2 Literature Review

As above mentioned, the main two properties that increase the computational complexity of the weaving scheduling problem are the job splitting and the sequence-dependent setup times. Both properties have been studied extensively under abstract models of various machine environments and optimisation criteria (Allahverdi et al., 2008; Rosales et al., 2015; Peyro, 2020; Lee et al., 2020; Peyro et al., 2019; Correa et al., 2016) and tackled through exact methods, approximation algorithms and metaheuristics. The weaving scheduling problem has also been well-studied and admits exact polynomial time algorithms for special cases where setup times are independent and job splitting is relaxed to preemption (Serafini, 1996), as well as MILP models and efficient metaheuristics for the general case (Eroglu and Ozmutlu, 2017; Eroglu et al., 2014; Rosales et al., 2015; Wang and Wang, 2015; Pimentel et al., 2011).

According to our knowledge, the most relevant previous work appears in (Lee et al., 2020; Peyro, 2020). Lee et al., (2020) proposed near optimal heuristics for a simplified model with identical machines, job splitting, multiple setup resources and fixed (independent) setup times per job, under the makespan minimisation objective. Peyro (2020) proposes a Benders Decomposition approach and heuristics for the general case of unrelated machines, sequence-dependent setup times and multiple setup resources, again under the makespan minimisation objective. However, none of these works combine all the complex properties needed for PIACENZA's case. Interestingly, Lee et al. (2020) referred to a case combining job-splitting, sequence-dependent setup times, unrelated machines and setup resource constraints as an open research direction.

4.3 Model and/or Solution method (Demonstration)

Let J be the set of jobs (orders), and M the set of machines (looms). Each machine m has a fixed speed s_m (in strokes/min) and each job i has a quantity v_i (in meters) of the fabric type that should be produced on one or more machines. Each fabric type is associated with a list of attributes such as

Model Parameters	
J	The set of jobs (orders)
M	The set of machines (looms)
s_m	The fixed speed (in strokes/min) of machine $m \in M$
q_i	The quantity (in meters) of job $i \in J$
u_i	The number of strokes/meter for the fabric type of job i,
$p_{i,m}$	The processing time of $i \in J$ on $m \in M$, $p_{i,m} = q \cdot u_i / s_m$
$S_{i,j,m}$	The setup time of $j \in J$ succeeding job $i \in J$ on $m \in M$
$\bar{S}_i$	The setup time of jobs processed first on each machine (1h)
L_i	A lower limit on the quantity of part of job $i \in J$ allocated to any machine (50 meters)
d_i	The deadline of job $i \in J$, i.e., a strict delivery date for i
$T_{\max}$	An upper bound on the makespan of an optimal schedule, e.g., $T_{\max} = \sum_{i \in J} \max_{m \in M} (p_{i,m} + \max_{j \in J} S_{i,j,m})$.
$\mathcal{T}$	A set of equal-length intervals $[\tau_{i-1}, \tau_i)$, $1 \le i \le T$, where $\tau_0 = 0$ and $\tau_T = T_{\max}$
$l_{i,j,m}$	The number of intervals needed to setup job j after job i on machine m
l_τ	The length of every interval $t \in \mathcal{T}$ (2h, which is the least common multiple over all setup times)
$\mathcal{D}$	A partition of $\mathcal{T}$ into subsets of consecutive time intervals q with total length equal to a working day
u_q	The allowed setup time per working day $q \in \mathcal{D}$ (50h)
R	A setup resource constraint to indicate that, at each time interval, at most R machines can be set up in parallel

Variables	
$X_{i,j,m,t}$	1 if $j \in J$ is the successor of $i \in J$ on machine $m \in M$, which is set up right after i at time $t \in \mathcal{T}$ and there are no other jobs processed between them on m, 0 otherwise
$X'_{i,m,t}$	1 if $i \in J$ is under setup on machine $m \in M$ at $t \in \mathcal{T}$, 0 otherwise
$Y_{i,m}$	1 if $i \in J$ is assigned on machine $m \in M$, 0 otherwise
$Q_{i,m} \in \mathbb{R}^+$	The quantity of job $i \in J$ to be processed by $m \in M$
$C_{i,m} \in \mathbb{R}^+$	The completion time of the part of job $i \in J$ processed on $m \in M$
$C_{max} \in \mathbb{R}^+$	The makespan of the schedule

Table 4. Model Parameters and Decision Variables

number of yarns, strokes per meter, “annotability” and “chainability” codes, comb height and code and complexity index, which can be used to calculate the processing time of each job i in machine m, namely $p_{i,m} = q \cdot u / s_m$, where q is the quantity of job i and u_i the number of strokes/meter for the fabric type of job i, as well as the setup time $S_{i,j,m}$ of a job j succeeding job i, $j \neq i$, on machine m. To present our mixed integer linear program (MILP), we summarize the notation in Table 4.

$$\textbf{(MILP)}: \quad C_{max}$$

s.t. :

$$\sum_{i,j\in J, i\neq j} X_{i,j,m,t} \leq 1, \qquad \forall t \in \mathcal{T}, m \in M \qquad (4.1)$$

$$\sum_{i\in J} X'_{i,m,t} \leq 1, \qquad \forall t \in \mathcal{T}, m \in M \qquad (4.2)$$

$$\sum_{t\in\mathcal{T}} X_{i,j,m,t} \leq 1, \qquad \forall i,j \in J, i \neq j, m \in M \qquad (4.3)$$

$$\sum_{j\in J, t\in\mathcal{T}} X_{0,j,m,t} \leq 1, \qquad \forall m \in M \qquad (4.4)$$

$$\sum_{m\in M} Q_{i,m} = q_i, \qquad \forall i \in J \qquad (4.5)$$

$$L_i \cdot Y_{i,m} \leq Q_{i,m} \leq q_i \cdot Y_{i,m}, \qquad \forall i \in J, m \in M \qquad (4.6)$$

$$Y_{i,m} = \sum_{t\in\mathcal{T}, j\in J, j\neq i} X_{i,j,m,t}, \qquad \forall i \in J, m \in M \qquad (4.7)$$

$$Y_{j,m} = \sum_{t\in\mathcal{T}, i\in J, i\neq j} X_{i,j,m,t}, \qquad \forall j \in J, m \in M \qquad (4.8)$$

$$\sum_{t\in\mathcal{T}} X_{i,j,m,t} + \sum_{t\in\mathcal{T}} X_{j,i,m,t} \leq 1 \qquad \forall i,j \in J, i,j \neq 0, i \neq j, m \in M \qquad (4.9)$$

$$\sum_{i,j\in J, i\neq j, t\in\mathcal{T}} X_{i,j,m,t} = \sum_{i\in J} Y_{i,m} - 1, \qquad \forall m \in M \qquad (4.10)$$

$$X_{i,j,m,t} + \sum_{\substack{i'\in J, i'\neq i, \\ t'\in\mathcal{T}, t'\leq t}} X_{j,i',m,t'} \leq 1, \qquad \forall i \in J, j \in J, i \neq j, m \in M, t \in \mathcal{T} \qquad (4.11)$$

$$X_{i,j,m,t} \cdot l_{i,j,m} \leq \sum_{t'=t}^{t+l_{i,j,m}-1} X'_{j,m,t'}$$

$$\forall i,j \in J, i \neq j, m \in M, t \in \mathcal{T} \setminus \{T - r | 1 \leq r \leq l_{i,j,m}\} \qquad (4.12)$$

$$\sum_{t\in\mathcal{T}} X'_{j,m,t} \leq \sum_{i\in J, i\neq j, t\in\mathcal{T}} l_{i,j,m} \cdot X_{i,j,m,t}, \qquad \forall j \in J, m \in M \qquad (4.13)$$

$$\sum_{i\in J, m\in M} X'_{i,m,t} \leq R \qquad \forall t \in \mathcal{T} \qquad (4.14)$$

$$\sum_{i\in J, m\in M, t\in q} l_\tau \cdot X'_{i,m,t} \leq u_q \qquad \forall q \in \mathcal{D} \qquad (4.15)$$

$$C_{j,m} - C_{i,m} + V(1 - \sum_{t\in\mathcal{T}} X_{i,j,m,t}) \geq Q_{j,m} \cdot \frac{u_j}{s_m} + S_{i,j,m} \cdot \sum_{t\in\mathcal{T}} X_{i,j,m,t},$$

$$\forall i,j \in J, j \neq i, m \in M \qquad (4.16)$$

$$C_{j,m} \geq \sum_{i\in J, i\neq j, t\in T} X_{i,j,m,t}(\tau_{t-1} + S_{i,j,m}) + Q_{j,m} \cdot \frac{u_j}{s_m}, \qquad \forall j \in J, m \in M \qquad (4.17)$$

$$\bar{S}_i \cdot Y_{i,m} + Q_{i,m} \cdot \frac{u_i}{s_m} \leq C_{i,m} \leq C_{\max} \qquad \forall i \in J, m \in M \qquad (4.18)$$

$$\sum_{j\in J, t\in\mathcal{T}, t>\lceil \frac{R}{M} \rceil} X_{0,j,m,t} = 0 \qquad \forall m \in M \qquad (4.19)$$

$$Y_{i,m}, X'_{i,m,t}, X_{i,j,m,t} \in \{0,1\}, C_{i,m}, Q_{i,m} \in \mathbb{R}^+, \qquad \forall i,j \in J, m \in M, t \in \mathcal{T}$$

The overall goal is to minimise the makespan of the schedule, denoted as C_{max}. Since setup times are strictly positive, it is easy to prove that each machine processes at most one part of each split job. We refer to the above problem as the *Weaving Scheduling* problem, which is *NP-hard* even if machines are identical, job setup times are fixed (and independent) and R=1 (Letsios et al., 2021).

(MILP) is partly inspired by formulations on special cases (Lee et al., 2020; Rosales et al., 2015), extending them to capture the elaborate structure of *Weaving Scheduling*. More specifically, Constraints (4.1)-(4.4), (4.7)-(4.11), (4.19), are used to ensure the feasibility of job assignment, respecting that each machine processes at most one single part of each split job. Constraints (4.5)-(4.6) allow for job splitting with respect to the quantity limits. Constraints (4.12), (4.13), (4.16) ensure that the setup of each job part precedes its execution on the corresponding machine and calculate its completion times. Constraints (4.14), (4.15) are setup resource constraints, and Constraints (4.17), (4.18) provide tight lower bounds.

4.3.1 Combinatorial Heuristics

Using an exact commercial solver (Gurobi 9.1) on (MILP), we can solve many daily instances (i.e., ones with orders arriving at the same date) in a few minutes either optimally or by a small gap. Hence (MILP) could be used to support short-term goals like scheduling jobs in a daily manner. However, to fully support the business needs of a weaving enterprise, including mid- and long-term goals, it is important to efficiently tackle larger real instances.

Next, we propose two combinatorial heuristics, GH1 and GH2, which differ in the way they handle job splitting and assignment of each part of a job to a machine, while handling the sequence-dependent setup time and setup resources in the same way. Table 5 summarizes the notation used in the present and the following section.

Notation	
GH1	The first greedy heuristic
max_assgn	Upper limit on job assignments in each iteration of GH1
GH2	The second greedy heuristic
LPT	Longest Processing Time first rule used in GH2
λ	A positive constant chosen on the assignment step of GH2
ld_m	The load of $m \in M$ on the assignment step of GH2
aTSP	The Asymmetric Traveling Salesman Problem
LB	A lower bound derived by the solution of the MILP in GH1
% Gap	The percentage gap of GH1 or GH2 wrt LB: $\frac{\{\text{GH1,GH2}\}-LB}{LB} \cdot 100$
Tardiness	For each job $j \in J$ in a schedule it is equal to $\max\{C_j - d_j, 0\}$
Tardy job	A job $j \in J$ with positive tardiness in a schedule, i.e., $C_j > d_j$

Table 5. Algorithms and experiments parameters and abreviations

GH1, performs an iterative exact splitting and assignment of jobs (parts) to machines using a MILP formulation (which is a subproblem of *Weaving Scheduling* where setup resource constraints are not taken into account) that minimises makespan subject to Constraints (4.5), (4.6) (to ensure that quantity limits are satisfied), (4.20) that calculates a lower bound on the time needed to process the assigned part of each job on each machine and (4.21) that limits the number of possible job assignments to max_assgn. GH1 starts by setting the maximum possible value of max_assgn = |J|•|M| and after each iteration decreases it by 1, to exploit all possible exact solutions (of increased or decreased job splitting potential) choosing the best among them. It terminates when the number of jobs exceeds the possible assignments i.e., max_assgn = |J| - 1, as there is no possibility to assign all jobs.

$$\sum_{i \in J}(Q_{i,m}\frac{u_i}{s_m} + Y_{i,m} \cdot \min_{j \in J} S_{i,j,m}) \leq C_{max} \qquad \forall m \in M \qquad (4.20)$$

$$\sum_{i \in J}\sum_{m \in M} Y_{i,m} \leq \max\_assgn, \qquad \forall m \in M \qquad (4.21)$$

On the other hand, GH2 performs a greedy job splitting dividing job quantities into parts equal to the lower bound L_i: For each job i with $q_i \geq 2L_i$ we create $\alpha = \left\lceil \frac{q_i}{L_i} \right\rceil$ job parts of quantity equal to $\frac{q_i}{\alpha}$. Then the job parts are ordered according to the LPT rule, to prevent the resulted schedule from unbalanced machine loads (i.e., when a job with large processing time is scheduled last). Then assignment process is similar to the one proposed by Aspnes et al. (1997) for makespan minimisation on unrelated processors: For the LPT order of job parts, it assigns each part i to the machine k = $argmin_{m \in M}\{\lambda^{ld_m + S_{j,i,m} + Q_{i,m}\frac{u_i}{s_m}} - \lambda^{ld_m}\}$ where j is the last job executed on m before i.

Both GH1 and GH2 are then following the next two stages. Stage A: For each machine, the assigned job parts are scheduled optimally by reducing the problem to aTSP, where nodes correspond to jobs' parts and the distance between nodes to sequence-dependent setup time plus processing time of the corresponding job part; the exact approach of Roberti and Toth (2012) is proved quite efficient for our instances. Stage B: For each machine in decreasing order of load and each available group of workers, we compute the earliest time that a job part can start its setup, respecting the order of job parts from Stage A. Note that, in Stage B, by starting from the most loaded machine, we significantly reduce the effect of idle intervals between consecutive job executions on the final makespan. Moreover, in the case of GH2, we do not violate the assumption that each machine processes at most one single part of each split job, as the setup time between parts of the same job is equal to zero, and thus in the aTSP solution they will be consequently ordered.

Summarizing, GH1 performs an exhaustive job splitting and assignment supported by an exact solver, while GH2 computes a fast greedy assignment of all possible job parts to machines.

4.4 Computational Experience

The experiments are performed on 27 weekly instances, from 01/2020 - 07/2020. The number of jobs per instance ranges from 7-69, the available groups of workers and number of machines per week are R=3 and 12 respectively, while setup times receive values from the set {2h,4h,6h}. The experiments ran on a 64-bit Windows PC (Intel i5, 2.5GHz CPU speed, 8GB RAM) using Python 3.7.2 for GH1, GH2 and GUROBI 9.1 (Python API for (MILP) and MILP of GH1).

We tested (MILP) on the above dataset, with a 2-hour limit, on 4, 6, 8 and 10 machines and it was able to solve optimally one weekly instance (7 orders) on 10 and 8 machines in 10 sec and 25 sec respectively, while the other two instances were solved with Gaps 8.62% after 262 sec for 6 machines and 5.14% after 1735 sec for 4 machines. The difficulty of (MILP) to deal with job splitting property lies on the fact that the time horizon (thus, the number of time intervals and the number of variables) increases exponentially as the quantity of the job increases.

Interestingly, the above results refer to the solution of Gurobi when using as upper bound the best among GH1 and GH2 solutions (normalizing processing times and setup times as multiples of l_τ, otherwise we could only handle some daily instances. So, we proceed by applying GH1 and GH2 to solve our weekly instances. To better evaluate the performance of GH1 and GH2 we divide our dataset into five subsets of increasing number of jobs, each consisting of 5-6 weekly instances, and we test each subset for different number of available machines (4,6,8,10,12).

	% Gaps of GH1					Mean	Mean	% Gaps of GH2					Mean	Mean
# Orders	4	6	8	10	12	Gap (%)	t(s)	4	6	8	10	12	Gap (%)	t(s)
[7, 26]	3.2	4.92	6.99	10.45	10.39	7.19	38.7	13.03	22.46	30.21	38.43	42.15	29.25	3.9
[35, 40]	2.2	3.21	4.2	5.67	5.99	4.25	34.9	11.07	16.94	22.91	36.8	29.82	23.51	25.3
[43, 45]	2.14	2.77	4.17	5.59	7.05	4.34	38.6	10.87	14.88	23.78	35.5	38.63	24.73	8.7
[46.51]	1.97	2.86	3.7	4.82	6.09	3.89	33.8	11.16	15.95	23.02	33.86	30.72	22.94	6.5
[57, 69]	1.76	2.54	3.37	4.39	5.62	3.54	36.1	12.98	17.96	22.7	31.22	24.05	21.78	15.9
Mean Gap	2.28	3.3	4.55	6.29	7.1	4.71	-	11.84	17.75	24.68	35.23	33.32	24.57	-
Mean t(s)	6.3	35.4	33.5	46.5	59.8	-	35	26.9	7	5.3	4	14.6	-	11.6

Table 6. Results over all weekly instances on 4, 6, 8, 10 and 12 machines

As we show in Table 6, GH1 outperforms GH2, achieving results of 4.2 times smaller gap, but being 3 times slower on average, over different numbers of available machines. Notably, GH1 achieves almost optimal solutions of Gap less than 7.1% (4.7% on average) for all instances, in less than a minute (35sec on average). Note that instances with a few orders on many machines seem to demonstrate larger Gaps, compared to smaller number of machines mainly due to the total setup time constraint and the limited number of groups of workers. Additionally, running times may seem inconsistent regarding the size of the instances, but this is justified due to the small number of instances of each subset. As a result, instances that are time-consuming within a subset have a huge impact on the average running time.

4.4.1 Enhancements

It is important to note that, due to the COVID-19 situation, 22.13% of orders were tardy. Even though the solutions in Section 4.1 achieve small gaps, they cause a significant increase on the number of tardy jobs which therefore rise to 27.4% of the total orders (an increase of 24% compared to the ones initially tardy).

Moreover, observing that small jobs (unsplittable with $q_i < 100$ meters) have tight deadlines, while larger ones have looser, it appeared reasonable to dedicate a set of machines to small jobs and the rest to the large ones. To this direction, we perform a comparison of GH1 and GH2 on small and large jobs separately, to decide which is the best choice in every case. We divide each weekly instance to small and large jobs and, as before, we divide our dataset into subsets of increasing number of (either small or large) jobs. Subsets with small jobs consist of 4-6 weekly instances each, while subsets of large jobs of 5-7; note that on the latter we have excluded two instances, since they included only 1 and 2 jobs, respectively. The size of small job instances ranges from 6 to 41, while for large from 5 to 34.

# Orders	% Gaps of GH1				% Gaps of GH2			
	4	6	8	10	4	6	8	10
[6, 12]	3.78	8.53	5.36	4.37	27.11	26.67	20.13	15.36
[14, 22]	6.64	9.55	16.69	19.84	22.66	41.52	44.42	52.07
[25, 29]	8.26	20.53	48.24	58.43	16.35	32.03	55.16	68.08
[30.31]	6.69	25.8	60.75	77.98	26.49	53.8	64.05	102.75
[33, 41]	6.41	16.33	45.5	65.08	15.13	23.86	51.83	56.02
Mean Gap	6.54	16.35	36.6	46.94	20.95	35.56	48.49	60.45
Mean t(s)	9	2.7	5.7	30.6	1.1	1.5	2	2.4

# Orders	% Gaps on GH1				% Gaps on GH2			
	4	6	8	10	4	6	8	10
[5, 12]	2.96	4.23	6.04	7.82	17.45	38.01	30.82	64.14
[13, 16]	1.66	2.46	3.4	4.71	10.14	20.66	24.61	46.55
[18, 19]	1.58	2.29	3.46	4.19	8.88	13.43	21.61	29.36
[20.34]	1.02	1.53	2.27	3.21	7.89	14.34	19.7	25.27
Mean Gap	1.78	2.59	3.73	4.93	11.01	21.61	24.12	41.38
Mean t(s)	9.9	358.1	208.6	300.1	125.8	51.7	13.8	11.5

Table 7. Results on small (left) and large (right) job instances for 4, 6, 8, 10 machines

We tested (MILP) on small jobs, using a simplified version (where in constraints (4.16)-(4.18) we substituted $Q_{i,m} \bullet \frac{u_i}{s_m}$ by $Y_{i,m} \bullet p_{i,m}$, while Constraints (4.5)-(4.6) were removed) on 4, 6, 8 and 10 machines, for 8 out of 27 weekly instances (from 6 to 18 orders). Notably the exact solver was able to solve optimally 20 instances in 98.03 sec on average, 10 instances were solved with mean Gap 7.36% and for 2 it was not able to obtain a solution under 1-hour limit. However, since the solutions obtained were of similar Gap with the ones of GH1, we do not present them in more detail. Table 7 presents the comparison between GH1 and GH2 on small and large job instances, respectively. GH1 achieves solutions of better quality, with 26.6% and 4.8% Gap for small and large jobs respectively, however GH2 is much faster (4 times on large and almost 6 times on small jobs). Interestingly, for small jobs the difference on their gap is significantly decreasing (from 410% on large jobs to 55%). Note that Gap values on small jobs instances are quite large, but this is due to the strict daily total setup time constraint.

Since GH1 performs better on both small and large job instances, we run it once to schedule first all small jobs to an appropriate number of machines and re-run it consequently to schedule the large jobs on the remaining machines or (if possible) after the small jobs on their dedicated machines. More precisely, we run GH1 for each weekly instance, for 12 candidate numbers of dedicated machines ($|M| \in \{1,2,\ldots,12\}$) on small jobs.

The aim of this approach is to examine the effect of dedicated machines on three optimisation criteria: makespan, number of tardy jobs and total tardiness.

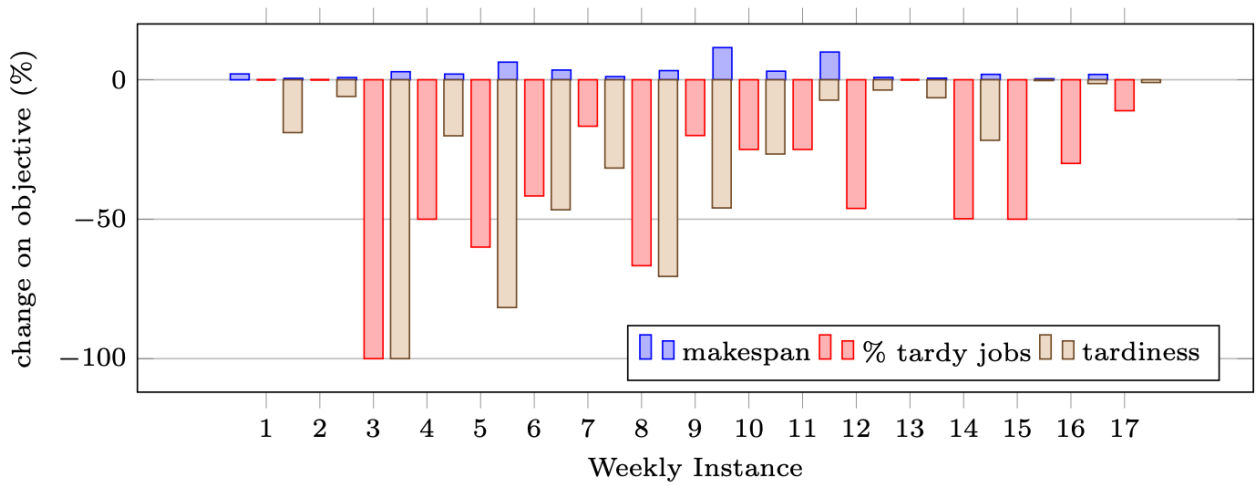

Figure 7. Best policies to balance makespan, number of tardy jobs and total tardiness, over weekly instances

We consider as baseline the makespan, number of tardy jobs and total tardiness over all weekly instances computed by GH1 in Section 4.4 and highlight the smallest average change on each criterion over the same instances, over all runs under different number of dedicated machines: For makespan, the smallest average increase is 1.55%, while for the same instances tardiness and the number of tardy jobs decrease by 16.68% and 10% respectively. For the number of tardy jobs, the largest average decrease is 16%, while for the same instances the makespan increases by 4.07%, and tardiness decreases by 19.1%. For total tardiness, the largest average decrease is 22.62%, while for the same instances makespan increases by 6.35% and number of tardy jobs decreases by 12.12%.

Figure 7 presents a proposed policy for weekly instances, to achieve better tradeoffs between makespan increase and number of tardy jobs, tardiness decrease. We conclude that dedicating machines on small jobs positively affects 17 out of 27 instances (in Figure 7), trading a small increase on makespan for large reductions on the number of tardy jobs and total tardiness. Notably all improvements occurred when the number of dedicated machines ranges from 2-7, while in 76\% of the instances the range is from 2-4. It is also encouraging that on 15 of those 17 instances there were various alternative policies that could be chosen demonstrating also positive effects.

Let us conclude by saying that additional experimentation, on both real and random or modified literature instances, could yield more insights. Although already competitive within a quite challenging setting, our optimisation approach will be further strengthened by examining tighter formulations in a combination with a Benders-like decomposition, to accomplish provably near-optimal solutions on even larger instances. Moreover, the robustness of our approach against common disturbances (such as loom malfunctions) will be also tested in the next stage of the project.

5 Automotive Manufacturing: Pilot Case by CONTINENTAL

5.1 Introduction

Continental is a discrete part manufacturing environment for automotive parts. Key decision areas for the operation managers and production line supervisors are the following:

a) generation of static schedules for a given set of production orders and available resources considering a planning horizon of multiple days and various operational constraints,
b) reactive re-scheduling of the master plan as new input information arrives (e.g. new urgent orders) based on the current status of the production lines, and
c) integrated scheduling of maintenance activities.

The production line examined in the context of FACTLOG consists of two stages. The first stage is preassembly subline that consists of 4 process steps, followed by one buffer step. Afterwards, the second stage contains 13 assembly process steps. Each process step is performed by one or more machines that can be seen as a workstation. All jobs follow the same routing through each line (there is no flexibility), which means that schedules can be determined at the level of the lines and not at the level of each individual workstation. No internal buffers are considered between workstations, and there is no parallel processing. Change of type of parts processed in the lines is followed by a setup time of one or more workstations. The transportation times for the movement of parts from one workstation to the next is considered negligible.

The above-described production setting can be modelled as a so-called flow shop scheduling problem (FSSP) with resource constraints (e.g. semi-finished products, raw materials etc). More specifically, all products follow a specific processing flow across multiple processing stages that may consist of one or multiple workstations. In a flow shop environment, there is a set of production orders (or jobs) that must be completed. Each order refers to a batch of products with similar characteristics and consists of several sequential operations that correspond to the processing of a job on every processing stage. The job size, resource consumptions, bill of materials and due dates are known. Each operation should be scheduled on specific workstation and no pre-emption / interruption is allowed. The processing times and the setup times (if needed) per operation on each workstation are known. The goal is to produce a schedule such that the completion time of the latest job is minimized (makespan) as well as the total job tardiness.

Apart from resource and other operational constraints, another important dimension is the scheduling of maintenance activities. It is very important to generate maintenance plans that will not create long delays on production orders with very tight deadlines and overall to minimize the negative impact of downtimes on the overall production schedule. Therefore, it is essential to treat the production and maintenance scheduling as integrated problems. Assuming that maintenance windows at specific machines are provided either from a predictive or preventive maintenance module, the aim is to schedule all production orders as well as to decide when the best time is to perform the machine maintenance activities.

A Constrained Programming (CP) approach is proposed for modelling and solving this integrated 2-stage Flow Shop Scheduling Problem with Resource Constraints and Maintenance Windows. The optimization model assumes an input the planning horizon, the

set of production orders to schedule, the available resources and the windows to perform maintenance activities. On return, it provides the optimal or near optimal production and maintenance schedules. On this basis, the core functionalities provided to the human planners are the following:

- ability to generate static plans and perform what-if scenarios (for example, the planner can generate plans for different sets of production orders, different resource availability, different processing times etc), and the
- ability to update the baseline master plan as dynamic events occur.

The dynamic events can be new urgent orders and/or machine breakdowns. Whenever the planner applies a dynamic event together with the baseline plan, a new re-optimized plan will be generated. As described earlier in this report, the optimization engine is a Restful API that accepts requests and delivers responses using Hypertext Transfer Protocol (HTTP) and JSON text format for data exchange. Appendix 3 presents in detail the inputs and outputs in terms of classes and data structures.

Below, Section 5.2 provides a brief overview of the literature on flow shop scheduling problems with resource constraints as well as on integrated shop scheduling problems with maintenance planning and scheduling. Section 5.3 describes in detail the optimization model. Finally, Section 5.4 provides some preliminary computational experiments on synthetic data sets.

5.2 Literature Review

There is a huge literature on shop scheduling problems and various exact and heuristic approaches have presented and tested on well-known benchmark data sets for a wide variety of problem variants with various mixes of constraints and (multi-)objective functions. We refer interested readers to the recent survey paper of Komaki et al. (2018) on Assembly Flow Shop Scheduling problems. Various papers also propose models and algorithms for problems with resource constraint; however, the literature in this domain is less organised.

Overall, there are 5 families of resources, namely renewable resources, non-renewable resources, work-in-progress buffers, bill of materials and tooling resources. The most common case of renewable resources are utility resources (e.g. electricity) that are consumed from the machines during their operation. Often both soft and hard limits are imposed on the usage of utility resources. Non-renewable resources are typically used to describe material resources that are consumed and/or produced during job operations. Work-in-Progress buffers are intermediate capacity buffers and describe constraints that exist before/after machines. These buffers are used to hold jobs when they cannot be directly processed from the next machine. Finally, tooling constraints are used to describe limited capacity renewable resources that are occupied by tasks during their execution, and they are freed once the processing finishes. In practice, this kind of resources can be used to describe expert personnel that is required to operate specific machines or to execute specific tasks, or special equipment that is limited in the shop floor.

Most shop scheduling problems studied in the literature assume unlimited capacities and work-in-progress buffers, and therefore, no waiting is imposed to the execution of any operation. By adjusting the size of buffers one can enforce the blocking of the execution of the operations, and hence, cause a dramatic increase to the makespan. Blocking constraints and limited capacity buffers for the Flow Shop Scheduling Problem (FSSP) appear in the

work of Trabelsi et al (2012). In this paper, a continuous production shop floor is assumed with multiple stages, while heuristic and metaheuristic algorithms are proposed for the so-called FSSP with mixed blocking constraints. Mascis and Pacciarelli (2002) uses the Job Shop Scheduling Problem (JSSP) to study blocking constraints imposed by zero capacity intermediate buffers. In a more generic fashion, Brucker et al. (2006) tries to organize the possible buffer options and also provides essential definitions and disjunctive graph modifications for a more efficient representation of the JSSP variant. Yaurima et al. (2009) studies a hybrid FJSSP problem with unrelated machines, sequence dependent setup times and limited buffers inspired by a television assembly shop floor. Lastly, Belaid et al (2012) study a two machine Flexible JSSP with limited capacity temporary buffers between production stages, inspired by a shampoo industry and provide heuristic and metaheuristic approaches for solving the problem. To our knowledge literature regarding blocking constraints on Flexible JSSP with parallel machines are very limited. Aschauer et al (2017, 2018) study Flexible JSSPs with no-wait constraints inspired by a hot rolling mill application, while Groflin et al (2011) develop a metaheuristic algorithm for a similar problem.

In recent years, shop scheduling integrated with maintenance planning and scheduling has received a lot of attention. Many different types of maintenance have been considered, including among others PM (Preventive Maintenance) RTFM (Run to failure maintenance) CBM (Condition-based maintenance), Corrective Maintenance (CM), TBPM (Time-based Preventive Maintenance) and RCM (Reliability centered Maintenance). In cases of preventive maintenance various stochastic aspects has been modelled and many different policies has been tested. Assuming a deterministic setting, one approach that seems to be effective is to consider a priori maintenance windows for fixed duration maintenance activities. These windows and the related breakdown probabilities can be derived via supervised machine learning models based on historical data. In addition, simulation models can be used to evaluate the generated schedules.

Table 8 provides a summary of the literature for integrated shop scheduling and maintenance planning.

Reference	Problem type	Maintenance type	Machine degradation	Notes	Objective	Stochastic aspect	Method
Zandieh et al (2017)	Flexible Job Shop	Basic & Preventive Maintenance	YES	Thresholds for the machine degradation dictate the type of maintenance activity. Schedules are evaluated using simulation	Makespan	Sigmanormal functions for maintenance duration. Sigmoid distributions for the shock events	Metaheuristic
Perez-Gonzales et al. (2020)	Flow Shop	Time-based Preventive Maintenance	NO	Resumable-non-resumable maintenance activities. Periodic and deterministic maintenance activities	Makespan, Lateness	-	Exact (MILP)
Branda et al. (2020)	Flow Shop	Preventive & Corrective Maintenance	NO	-	Makespan, Earliness-Tardiness	Randomized failure time of a machine that follows Weibull distribution	Genetic Algorithm
Dong et al. (2020)	Job Shop	Preventive Maintenance	NO	Fixed and Flexible maintenance activities. Single machine	Makespan, Total Flow Time	-	Exact (MILP)

Reference	Problem type	Maintenance type	Machine degradation	Notes	Objective	Stochastic aspect	Method
Shijin and Jianbo (2010)	Flexible Job-Shop Scheduling	Preventive Maintenance	NO	Deterministic maintenance time windows. Two types of resources are incorporated to constraint the ability of maintaining more than one machine simultaneously.	Makespan, Total workload, Critical machine workload	-	Filtered beam search
Cui et al. (2017)	Job Shop	Time based Preventive Maintenance	NO	Resumable-non-resumable maintenance activities	Makespan	-	Branch and Bound + Heuristic
Hadi and Mehrdad (2015)	Flexible Job Shop	Preventive Maintenance	YES	Discrete failure rates. Maintenance time is constant. Minimum availability constraint	Number of tardy jobs	-	SA + Monte Carlo Simulator
Azadeh et al (2015)	Open Shop	Preventive Maintenance	NO	-	Multiple (Makespan, Total Tardiness, Earliness, Machine availability)	Poisson distribution is used to calculate the time required for preventive maintenance.	MOPSO + NSGA II metaheuristics
Moradi et al (2010)		Preventive Maintenance	NO	Fixed maintenance activities on specific time periods. Everything else seems deterministic	Makespan	-	Preventive maintenance and learnable genetic architecture
Gholami et al (2009)	Hybrid Flow shop	Preventive Maintenance	YES	Machines suffer only breakdown events with stochastic intervals.	Makespan	Exp-rand function is used to calculate breakdown intervals and breakdown times.	Random key genetic algorithm
Naderi et al (2009)	Job shop	Preventive Maintenance	NO	Various maintenance policies	Makespan	-	Genetic Algorithm and Simulated Annealing
Ehram et al (2010)	Flow Shop	Preventive Maintenance	YES	Thresholds for machine degradation level. Metaheuristic is used for generating schedules that are evaluated using a simulator	Makespan	Shock events follow a poisson distribution, amount of degradation follows an exponential distribution, recovery time follows lognormal distribution	Hybrid simulated annealing-tabu search
Yu and Hee (2021)	Flow Shop	Preventive Maintenance	NO	Proportional Processing times	Total Completion Time, Maximum Lateness	-	Exact (MILP)

Reference	Problem type	Maintenance type	Machine degradation	Notes	Objective	Stochastic aspect	Method
Logendran and Talkington (1997)	Job shop with parallel machines	Preventive and Corrective Maintenance	YES	Two maintenance policies. Schedules are simulated		Mean time for machine failures follows an erlang distribution. Repair times are also within a range of values	-
Ruiz-Torres et al (2017)	Job Shop	Repair Maintenance	YES*	Deteriorating processing times after each job. Maintenance activities restore machine performance	Makespan	-	Heuristics
Shijin and Ming (2014)	Two-stage hybrid flow shop	Preventive Maintenance	YES	Start times of preventive maintenance activities are unknown as well as their number. Durations are fixed, availability uses a distribution.	Bi-objective (Makespan + Machine availability)	-	MOPSO + NSGA II metaheuristics
Bajestani et al (2014)	Flow Shop	Preventive Maintenance	YES	Machine deterioration states are defined, and the transitions follow markov chain rules	Maintenance cost + lost production cost due to late orders	Random-based transitions between machine states	MDP for the maintenance plan and MIP for the production scheduling
Ben Ali et al (2011)	Job-shop scheduling	Preventive and Corrective Maintenance	YES	Maintenance is applied based on 2 types of tasks (periodic and workflow based)	Multiple (Makespan, Total maintenance cost)	-	Genetic Algorithm
Rajkumar et al (2010)	Flexible Job Shop	Preventive Maintenance	NO	Start and end times of maintenance activities as decision variables	Weighted sum function of makespan, workload, total workload	-	GRASP
Yahong et al (2014)	Flexible Job Shop	Preventive Maintenance	NO	Maintenance time windows	Makespan	-	Heuristics
Rahmati et al (2018)	Flexible Job Shop	Preventive and Corrective Maintenance	YES	Machine status is checked on specific intervals, maintenance actions can be preventive or corrective. Thresholds control the availability of the machine. Schedule is evaluated through simulation.	Multi-objective (Makespan, maintenance cost function, system reliability function)	Shock events are stochastically applied and degrade the status of the machine. PM/CM activity durations are also stochastically calculated	4 multi-objective simulation based optimization algorithms (MOBBO, PESA, NSGAIII, and MOEAD)

Table 8: Literature review for integrated shop scheduling and maintenance planning

5.3 Model and/or Solution method (Demonstration)

5.3.1 Notation

The examined 2-stage FSSP with resource constraints and maintenance activities is modelled as follows. Let a set of jobs $J = 1, \dots, l$, set of available machines $M = \{1, \dots, m\}$, a set of tools $T = \{1, \dots, L_T\}$, a set of utility resources $U = \{1, \dots, L_U\}$, a set of arbitrary resources $R = \{1, \dots, L_R\}$ and a set of WIP Buffers $W = \{1, \dots, L_W\}$. We define two dummy operations i_u° and i_u^* for each job $u \in J$, which correspond to the first and the last operations of the job, respectively. Each job u consists of a set of operations O_u, including the dummy operations. There exists a set Ω that includes all the operations of the problem, $\Omega = \bigcup_{u=1}^{l} O_u$. Let $n = |\Omega|$ denote the total number of operations. Each operation $i \in \Omega$ can be executed on a set of available machines $M_i \subseteq M$ and has a processing time $p_{i,k}$, where $k \in M_i$. Each operation is executed once by a single machine, the machines can execute only one operation at a time and no pre-emption is allowed. During the execution time that machines execute operations, they may consume more than one utility resource. The flexibility fx of the problem can be defined as a metric of the degrees of freedom regarding the assignment of operations to different machines, and it can be calculated as $\frac{1}{n}\sum_{i=0}^{n}|M_i|$.

Each operation $i \in \Omega$ can be associated with two resources $Req(i)$ and $Prod(i)$ that correspond to the resources required and produced by the operation respectively, while the tool associated to the operation is denoted by $t(i)$. Note that in cases where there are no required/produced resources or a needed tool for an operation i, the values of the corresponding association vectors are set to -1. To reduce the complexity of the problem we assume that the produced and/or required resource quantity per operation is fixed ($lotSize$). Each utility $U_k \in U$ has a hard consumption limit denoted by $\overline{U_k}$. For a machine $k \in M\$, \$u\_i(k)$ is a binary variable that depicts whether or not machine k requires utility U_i. For simplicity we assume that each machine exhibits a unitary consumption per utility during each operation. The size of the limited capacity buffer of machine $k \in M$ is denoted by $lcb(k)$. For each tool $k \in T$ has a hard upper bound is defined, denoted by $\overline{T_k}$, that corresponds to the maximum number of instances of the tool that can be used in parallel. Starting inventory of a resource $k \in R$ is denoted by R_k^{start}. Finally, the associated work in progress buffer of a resource $k \in R$ is denoted by $wip(k)$.

Definition A. A solution s is defined as a pair (α, π), where α is a vector that represents the assignment information of operations to machines and π is a table of vectors that represents the sequence of operations executed at each machine.

More specifically, let $\alpha = \{\alpha(i), \forall i \in \Omega\}$, where $\alpha(i) \in M_i$, and $\pi = \{\pi_k, \forall k \in M\}$, where π_k denotes the permutation of operations processed by machine k. For the sake of completion, every permutation π_k starts and ends with two dummy operations $m_k^\circ, m_k^* \in$ that denote the start and the end operations of machine k, respectively. Note that $M_{m_k^\circ} = M_{m_k^*} = \{k\}$ and $p_{m_k^\circ,k} = p_{m_k^*,k} = 0$, for all $k \in M$. Note that given π, one can derive the assignment vector α, but for the sake of simplicity α is also included in the definition of a solution.

We use pm_i (and sm_i) to denote the machine predecessor (and successor) of operation i assigned to machine $\alpha(i)$ in a solution $s(\alpha, \pi)$. In the same manner, we use pj_i (and sj_i) to denote the single job predecessor (and successor) of operation i.

Definition B. The cost of a solution s, namely the makespan of the schedule C^s_{max}, is defined as the maximum completion time of all operations in Ω.

Definition C. The head times r_i denote the difference between the start time of the schedule and the start time of an operation i.

Definition D. The tail times q_i denote the difference between the completion time C_i of the operation i and the makespan C_{max}, i.e., $q_{C_{max}} - C$.

The head and tail times can be determined as follows:

$$r_i = \max\left(r_e + p_{e,\alpha(e)}, r_{pm_i} + p_{pm_i,\alpha(i)}\right) \forall i \in \Omega, e = pj_i \quad (5.1)$$

$$q_i = \max\left(q_e + p_{e,\alpha(e)}, q_{sm_i} + p_{sm_i,\alpha(i)}\right) \forall i \in \Omega, e = sj_i \quad (5.2)$$

Within the Flexible JSSP context critical components can be defined. The following definitions provide description for these concepts as well as the necessary notation.

Definition E. An operation i is critical when $C_{max} = r_i + p_{i,\alpha(i)} + q_i$.

In other words, critical operations have no flexibility to move back and forth in the scheduling horizon, and thus they define the length of the schedule, i.e., the makespan.

Definition F. A sequence of consecutive operations $B = \{o_1, o_2, \dots, o_{e-1}, o_e\} \subseteq \pi_k$ processed on the same machine k is considered as a critical block if all operations $i \in B$ are critical and $|B| \geq 2$.

In the following we provide the necessary definitions to help us solidify the concept of resource constraints as well as the different types of resources.

Definition G. A sequence of consecutive operations $B = \{o_1, o_2, \dots, o_{e-1}, o_e\} \subseteq \pi_k$ processed on the same machine k is considered as a critical block if all operations $i \in B$ are critical and $|B| \geq 2$.

5.3.2 Constraint Programming Formulation

Constraint Programming has been successfully applied for solving various highly constrained and large-scale scheduling problems. We refer interested readers to the works of Goel et al (2015), Rasmussen et al (2017), and Unsal and Oguz (2013). The input of a CP model is a set of decision variables, a finite set of alternative values as a domain per decision variable and a set of constraints that must be satisfied. A CP solver works by enumerating feasible solutions of the problem using branching algorithms. During this process, it also tries to decrease the domain cardinality of each decision variable by propagating through the constraints. Constraint propagation identifies values or combinations of values across multiple decision variables that cannot be part of a feasible solution, and therefore, can be excluded from the domain sets of the corresponding decision variables, which can lead to branch pruning (Laborie et al., 2018).

Specifically, for scheduling applications CP models use interval variables. This type of variable is a natural way of describing a task or activity. Interval variables have four

attributes: $IsPresent$, $Start$, End and $Size$. $IsPresent$ indicates if the interval variable is included in the solution or not, $Start$ and End denote the start and the end time of the interval variable, i.e., the start and the end time of the task, while $Size$ refers to the size of the interval, i.e., the length of the task.

In the CP Optimizer the notion of sequence interval variables is also defined, which are sets of interval variables that represent an ordering of the included interval variables. Specific constraints are also introduced by the $CP\ Optimizer$ to handle sequence interval variables. In our implementation the following constraints regarding sequence interval variables are used:

- $Before(a, b, c)$, within a sequence variable a, interval variable b should end before c starts.

In the following we include all expressions and functions used to deduce the status of an interval variable in the working solution of CP.

- $PresenceOf(a)$, is a boolean expression that returns the presence status of an interval variable a in the solution.
- $StartOf(a)$, is an integer expression that returns the start time of an interval variable a in the solution.
- $EndOf(a)$, is an integer expression that returns the end time of an interval variable a in the solution

To further simplify the modelling of resources, we again adopt the notation used by the ILOG CP Optimizer. More specifically, the CP Optimizer uses the notion of cumulative function expressions to model discrete cumulative functions over time. The CP Optimizer introduces several constraints on interval variables as well as the cumulative function expression themselves, to describe the contribution of each variable but also any constraints regarding the values of the cumulative function itself over specific time intervals. In the CP model implemented in this work, the following constraints are used:

- $Pulse(a, h)$, i.e., an interval variable a contributes h to the corresponding cumulative function during the execution time window of a
- $StepAtStart(a, h)$, i.e., an interval variable a issues a permanent contribution of h to the corresponding cumulative function at the start of a
- $StepAtEnd(a, h)$, i.e., an interval variable a issues a permanent contribution of h to the corresponding cumulative function at the end of a

In our implementation, for each operation i a decision interval variable τ_i is defined. The alternative execution options (modes) of an operation i on a machine $k \in M_i$ are also defined as decision interval variables $\phi_{i,k}$. For these variables a constraint is defined such that the $Size$ attribute of each $\phi_{i,k}$ is equal to the processing time $p_{i,k}$ of i on machine k. To accurately calculate the waiting times within the limited capacity buffers within the machines, for every available mode of an operation i on a machine $k \in M_i$, another decision interval variable $\phi_{i,k}^b$ is defined. For the sake of completion, we define a set $\mu_i = \{\phi_{i,k}, \forall k \in M_i\}$ to represent all the available execution modes per operation i, which is also used to denote the domain set of variable τ_i. Lastly, a sequence interval decision variable σ_k is defined per machine k over the set of interval variables $\sigma_k = \{\phi_{i,k}, \forall i \in \Omega\}$.

$$\min C_{max} \tag{5.3}$$

subject to:

$$Alternative(\tau_i, \mu_i) \forall i \in \tag{5.4}$$

$$EndBeforeStart(j, i) \forall i \in \Omega, \forall j \in PJ_i \tag{5.5}$$

$$NoOverlap(\sigma_k) \forall k \in M \tag{5.6}$$

$$\sum_{j=1}^{n} Pulse\left(\phi_{j,k}, u_i(k)\right) \forall k \in M_j \leq \overline{U_i} \forall i \in U \tag{5.7}$$

$$\sum_{j=1}^{n} Pulse\left(\tau_j, t_i(j)\right) \leq \overline{T_i} \forall i \in T, t_i(j) = i \tag{5.8}$$

$$\sum_{k=1}^{m} Pulse\left(\phi_{j,k}^{b}, 1\right) \leq lcb(k) \forall j \in M_j \tag{5.9}$$

$$PrecenseOf\left(\phi_{j,k}^{b}\right) = PresenceOf\left(\phi_{j,k}\right) \forall j \in \Omega, \forall k \in M_j \tag{5.10}$$

$$StartOf\left(\phi_{j,k}^{b}\right) = EndOf\left(\phi_{j,k}\right) \forall j \in \Omega, \forall k \in M_j \tag{5.11}$$

$$EndOf\left(\phi_{j,k}^{b}\right) = PresenceOf\left(\phi_{j,k}^{b}\right) StartOf\left(\tau_{sj_j}\right) \forall j \in \Omega, \forall k \in M_j \tag{5.12}$$

$$Step(0, R_i^{start}) + \sum_{j \in \Omega | Prod(j)=i} StepAtEnd\left(\tau_j, lotSize\right) - \sum_{j \in \Omega | Req(j)=i} StepAtStart\left(\tau_j, lotSize\right) \geq 0 \; \forall i \in R \tag{5.13}$$

$$\sum_{i \in R | wip(i)=k} \left\{Step(0, R_i^{start}) + \sum_{j \in \Omega | Prod(j)=i} StepAtEnd\left(\tau_j, lotSize\right) - \sum_{j \in \Omega | Req(j)=i} StepAtStart\left(\tau_j, lotSize\right)\right\} \leq \overline{W_k} \forall k \in W \tag{5.14}$$

$$C_{max} \geq EndOf(\tau_i) \forall i \in \tag{5.15}$$

The objective (5.3) refers to the minimization of the makespan. Constraints (5.4) are used to enforce a unique selection of the available modes for the interval variable τ_i out of the set μ_i. Constraints (5.5) are used to cover the precedence relations of the problem, i.e., each operation i can start as soon its job predecessor pj_i has finished. Constraints (5.6) ensure that the interval variables included in σ_k do not overlap, since a machine can execute only one operation at a time. They also ensure that each operation starts after its machine predecessor has finished. Constraints (5.7) and (5.8) are used to accumulate the consumption of utility and tool resources respectively. They also make sure that the upper usage bounds are not surpassed. Constraints (5.9, 5.10, 5.11, 5.12) are used to describe the usage of limited capacity buffers. More specifically, constraints 5.9 accumulate the usage of the limited capacity buffer and also impose the buffer capacity, while constraints 5.10, 5.11 and 5.12 are used to calculate the start and end times of the decision interval variables related to the limited capacity buffers. Constraints (5.13) are used to accumulate the production and consumption of each generalized resource, while constraints (5.14)

accumulate the usage of resources on their corresponding work in progress buffer. Lastly, constraint (5.15) is responsible for the calculation of the makespan.

In the above model, maintenance activities are added as additional dummy jobs / production orders with predefined release and due dates (to represent the maintenance window). These dummy jobs have zero processing time on all machines / workstations, except the one that maintenance will be performed.

5.4 Computational Experience

First, we assess the impact of resources constraints. we are using as a test bed for our experiments benchmark data sets for the generic Flexible Job Shop Scheduling Problem that is a generalization of the 2-stage FSSP. For this purpose, a subset of instances of the Fattahi dataset was chosen (MFJS1 – MFJS10). A hierarchical optimization objective was selected with the makespan as the primary objective and the maximum flow time as the secondary objective. A single utility resource is considered, that can limit the simultaneous operation of machines of the shop floor. The experiment is conducted in two steps. At first, an unlimited availability of the resource is considered. In this case, all machines can operate simultaneously without any restrictions or blockers. The CP model solved all problems optimally and the results for both objectives are presented in Column (RC_0) of Table 9. In the second step of the experiment, a restriction on the maximum allowed consumption of the resource is applied. The maximal resource limit is defined as a linear function of the number of available machines, so that the resources availability scales uniformly across all problem instances of the dataset. In this case, the CP model also managed to optimally solve all problem instances. The results are presented in Column (RC_1) of Table 9.

Instances				RC_0		RC_1		Impact $(RC_1-RC_0)/RC_0$	
#	N	M	Ops.	C_{max}	F_t	C_{max}	F_t	C_{max}	F_t
MFJS1	5	6	15	468	2054	805	3500	72%	70%
MFJS2	5	7	15	459	2072	803	2899	75%	40%
MFJS3	6	7	18	466	2501	996	5018	114%	101%
MFJS4	7	7	21	554	3352	1253	7126	126%	113%
MFJS5	7	7	21	514	3155	1191	6930	132%	120%
MFJS6	8	7	24	634	4212	1498	9636	136%	129%
MFJS7	8	7	24	879	5912	2051	13142	133%	122%
MFJS8	9	8	36	884	6753	2311	18015	161%	167%
MFJS9	11	8	44	1055	9316	2953	29368	180%	215%
MFJS10	12	8	48	1196	11575	3425	33295	186%	188%

Table 9: Fattahi Dataset with Resource Constraints (1 Resource + Hierarchical objectives Cmax | Ft)

The last Column of Table 9 provides the % increase of both objectives when considering limitations of resource constraints. The results show that in problems with the same number of machines, both objectives increase as the number of operations increases. The same effect is observed when the problem size increases (number of jobs as well as the number of operations). Overall, we notice that even a slight limitation of the maximal resource consumption limit (almost 20% across all problem instances), can cause significant increase to the makespan as well as the maximum flow time that can range from 70% to 190%. This

highlights the importance of applying exact optimization algorithms for production scheduling at assembly flow shops with resource constraints.

In addition to above sets of experiments, we also tested the scalability and efficiency of the proposed CP model on very hard-to-solve Flexible Job Shop Scheduling problems. For this purpose, we used various data sets from the literature. Table 10 summarizes the results obtained on small- and large-scale problem instances. Clearly, the CP models (using IBM ILOG CP Optimizer) performs exceptionally well on most common instances of the FJSSP. It managed to match 180 optimal solutions out of a total of 252 problem instances. Additionally, it manages to update 49 lower bounds out a subset of 178 instances, while also recording a total of 14 new best solutions. A recorded average gap of 1.54% shows that the CP model is able to calculate near optimal solutions within the time limit (3 hours in this case).

Benchmark Set	Number of problem instances	Number of operations	Number of machines / workstations	Avg. Cmax	Gap (%)
BRData	10	60-300	2-8	284.6	5.87
HURData	15	15-75	5-15	1428.3	0.96
HUVData	15	15-75	5-15	1366.0	0.07
HUEData	15	15-75	5-15	1697.4	0.47
CBData	21	100-225	11-18	995.2	0.00
DPData	4	12-56	5-10	2212.1	1.87
Average Gap				1.54	
# Optimal Solutions				180	
# New Best Solutions				14	

Table 10: Results on small and large sacle Flexible Job Shop Scheduling Problems

6 Steel Manufacturing: Pilot Case by BRC

6.1 Introduction

Scheduling is among the most important issues that concern the operation of manufacturing systems. Its aim is the efficient allocation of tasks to machines along with the subsequent time-phasing of this allocation. In general, tasks individually compete for resources which can be of a very different nature, e.g., manpower, money, processors (machines), energy, tools. The same is true for task characteristics, e.g., set up times, due dates, relative urgency weights, and functions describing task processing in relation to allotted resources. Moreover, a structure of a set of tasks, reflecting relations among them, can be defined in different ways. In addition, different criteria which measure the quality of the performance of a set of tasks can be considered (Blazewicz et al., 2014).

In this part we discuss *flow shop* scheduling problems, and more precisely we analyse the case of BRC Ltd, which is among the leading UK companies in steel reinforcement. Steel industry is an important industrial sector in UK and one of the biggest worldwide.

In what follows we briefly introduce the case of BRC, we describe its products, the manufacturing line and its major components, the warehouse procedure etc. Then we outline the relevant operational research literature.

Our aim is to develop a conceptual model for a part of the production after the "storage" and prior to "loading and dispatch" to the customers. We will construct a multistage flexible flow shop model and we will propose a suitable mixed integer programming model. Finally, we will present some preliminary results from the application of the MIP model on a set of randomly generated instances.

6.1.1 Scheduling

Today, resource management is an inevitable part of the performance and efficiency optimization in manufacturing and service industries. *Scheduling* is the allocation of shared resources over time to competing activities. It has been the subject of significant amount of research in the operations research field. Emphasis is given on investigating machine scheduling problems where jobs represent activities and machines represent resources; each machine can process at most one job at a time. The resources include the use of equipment, the utilization of raw material or intermediates, the employments of operators, etc. The purpose of scheduling is to optimally allocate the limited resources to processing tasks over time and the decisions to be determined include the optimal sequence of tasks taking place in each machine, the amount of material being processed at each time in each machine and sometimes the processing time of each job in each machine.

In addition, scheduling problems could be classified into *offline* and *online*. In an offline problem, the number of jobs, release dates, delivery dates, processing times, due dates and other input data are known in advance. When data are not known in advance, but they are realised only when a job is released then the problem is classified under the label of online scheduling. Such problems have been extensively used for resource planning in distributed systems (Hsu et al., 2010; Steiger et al., 2003.)

The two most common types of scheduling problems, which are native to manufacturing jobs, are *Job-Shop Scheduling Problem (JSSP)* and *Flow-Shop Scheduling Problem (FSSP)*. An important classification is based on the nature of the production facility to manufacture the required number of products utilizing a limited set of units. If production orders follow different production routes (require different sequences of tasks) and some orders may even visit a given unit several times it is known as a multipurpose plant and the related optimization problems are also called job-shop problems. If every job consists of the same set of tasks that are performed in the same order and the units are accordingly arranged in production line, it is classified as a multiproduct plant called flow-shop problem (Li and Ierapetritou, 2007). The latter class of problems is the ones that are mostly met in practise.

The main distinction between flow-shop and job-shop is that, in the former case each job passes the machines in the same order whereas in the latter case the machine order may vary per job. So, the arrival of a job at a particular machine is not stochastic and most of the jobs that flow through that machine are similar in nature. Since workflow in a job shop in not unidirectional, scheduling becomes quite harder and tedious. Jobs in a FSP are produced either continuously or in batches (Mahale, 2017). We consider the batch process in the sense that once processing of a batch is started, it cannot be interrupted, and other jobs cannot be introduced into the batch.

6.1.2 Case Description for BRC Ltd

BRC ltd is the UK's largest supplier of steel reinforcement and associated products for concrete. They fabricate cut \& bent rebar to the specs of BS8666:2005 and governed by the independent steel reinforcement governing body C.A.R.E.S. In 2009 BRC was acquired by the Celsa Steel Services UK group and currently has 4 depots in the UK with the largest being in Newport South Wales which can produce up to 2000 tonnes of fabricated reinforcement for the construction industry per week. The rest are in Romsey near Southampton, Mansfield in the midlands and Newhouse up in Scotland. BRC manufactures bespoke products for the construction industry with a lead-time of 5-7 days where each batch is unique and can be up to 2 tonnes of steel in one product batch. These can be in the form of simple straight bar, "U" shaped bars to complicated 99 shape codes where it could be 3D shapes. The process is to cut and shape from stock lengths of straight or coiled rebar and go through the flow process which will be explained with more details below.

Production transforms the stock into products which are placed by cranes in the finished product lay-down area. The orders (batches) are fulfilled by placing the various finished products, which these orders are composed of, onto the trailers. At this phase there is a scanning procedure where each product gets a time stamp. When the order is complete the batch is ready for shipment to customer. All the material movements inside the production line are made by cranes which are a limited shared resource. BRC reported that considering an additional crane is not an option due to space limitations.

We can segment the BRC factory into distinctive parts where different processes take place. Looking at figure (Figure 8) that displays the factory layout we see that it is segmented vertically into the left and right part responsible to produce *coils* and *bars* respectively. Additionally, distinct places are:

- A: Stock Coil (left) and Stock Bars (right) is stored in different places relevant to diameter.

- B: For the 3 bays different cranes transfer the raw material from A to any other of the three B's in order to always have stock to feed in the machines. When a crane operator observes that in any B there is a shortage of raw material, either coils or bars, he/she proceeds to move to A, pick up respective raw material and deposit to respective B.
- C: For the 3 bays this place denotes the position where the trailers that will carry the final products to the end customer(s) are located in order to be filled with completed products towards forming completed orders.
- D: Once a C is considered as full it proceeds to location D to be ready to leave the factory towards delivery to the end customer.

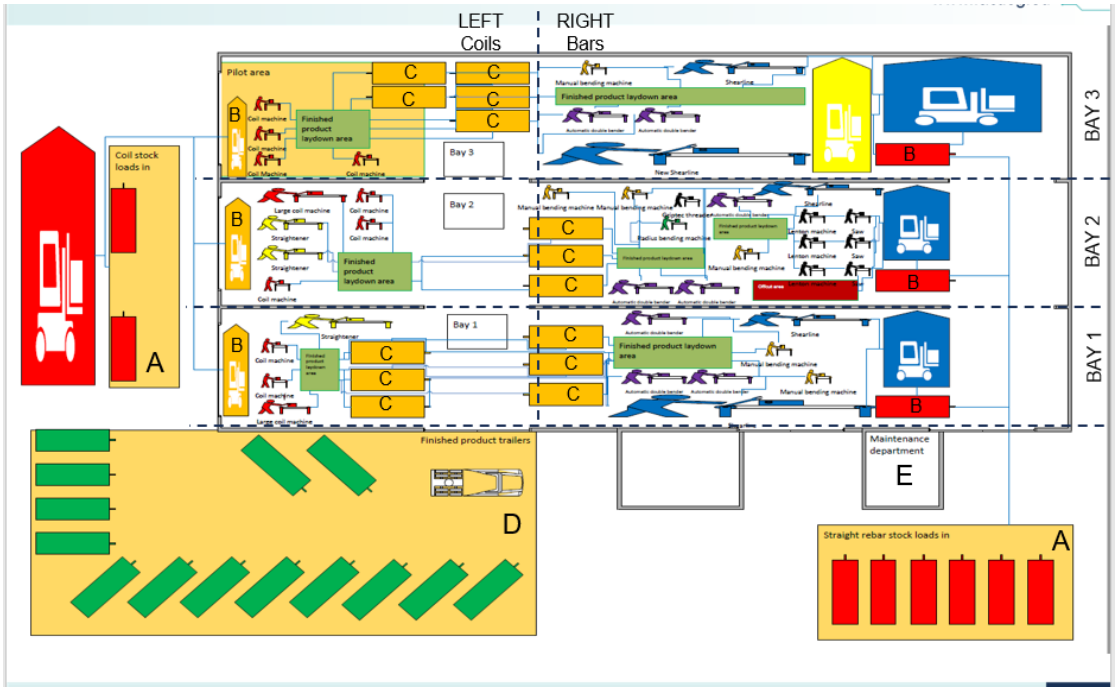

Figure 8. BRC Facility Layout

After receiving the raw materials, the company stocks them into its warehouse in bars or in coils. When an order is received, stock availability is considered. The coil material goes through bending in different shapes (it depends on the product code) or straightening and then is cut to length. The final products are temporarily placed to the finished product area and finally are loaded and transferred to the customers.

On the other hand, bars potentially need to go through a two-stage production process. A bar can be cut to length on shearlines or be dispatched as mill lengths regarding the order. The next job after cutting in size is either to dispatch the bar for shipping or threading and coupling. After finishing the latter procedure, the bars can be shipped or proceed to the final stage that is of bending. Finally, the products are temporarily placed to the finished product area and finally are loaded and transferred to the customers.

In the above figure there are three different types of trailers like Red, Yellow and Green, in a sense they correspond to the production cycle of BRC. The stock is piled in the Red trailers. Production transforms the stock into products which are placed by cranes in the “finished product area” (green block in the layout diagram). In the yellow trailers, the various

orders (batches) are fulfilled by placing the various finished products that these products are composed of. Scanning takes place at this phase. This means the product is given a time that was scanned in the yellow trailers. When the order is complete then the trailer becomes green, which means that the order can be shipped.

Although the company has a substantial processing capacity there is lack of system to organise the production plan. In the current processing system, the operator of each crane has a list of products that need to be moved but not an "optimised" order to do that. The principle in general suggests placing at the bottom pf the lay down area the straight bars, bent items are going next and small links at the very top. Since there is no picking system, the positioning and tracking of products/orders is rather problematic. While crane operators are looking for some products, they move other finished products around. As a result, a product might be under a lot of items when the operator is trying to locate it and that causes major delays.

The machines' idle is mainly caused by the delays of cranes and the lack of feasible schedules. The processing of a product might have finished in some station but there might be a further delay due to the shortage of cranes. Thus, the machine remains idle at this point. Another issue is the lack of data about the time that a crane needs to move a product inside the production. BRC will install some sensors to give time stamps when the crane collects an item.

We focus on bay 3, and more precisely on the flow shop scheduling problems for both coil and bar area. Given the orders in a specific time horizon, our goal is to find the optimal schedule with respect to specific *Key Performance Indicators (KPIs)*. In our case we will try to minimize the makespan C_{max} and the number of tardy jobs T_i. Those KPIs are encoded as follows:

- **Makespan (C_{max})**: one of the most common objective criterion. Makespan is the maximal (or latest) completion time of any job. The makespan is defined as max (C_1,...,C_n) where C_i is the completion time on the last machine for job i. With this goal the optimization method tries to finish each job as soon as possible. A minimum makespan usually implies a good utilization of the machine(s).
- **Number of tardy jobs (T_i)**: The number of tardy jobs is a measure that is quite often whether the company has very tight due dates in compare with the release times. The difference between the tardiness and the lateness lies in the fact that the tardiness never is negative. If the company allow tardy jobs after paying "something like a penalty e.g., complaints by the customers or a clause" then the model is more flexible but also more complex from computing time perspective since there are more possible combinations.

6.2 Literature Review

Over the last fifty years a considerable amount of research effort has been focused on deterministic and stochastic scheduling. In our case we will focus on deterministic Flow Shop problems. The number and variety of models considered is astounding. The FSP is one of the most complex scheduling problems and finding an optimal solution for real size instances in a reasonable amount of time is difficult both in practical and theoretical terms.

The main reasons that increase the computational complexity are the tardiness tolerance and the scale of the problem. Sometimes the company has some tight orders' due date so it's inevitable to avoid the job tardiness. In the above case our objective is to minimize as much as possible the number of tardy jobs or the convex combination with the makespan criterion. Flow shop problems have been studied extensively under exact and/or approximation methods using heuristics and metaheuristics with a variety of optimization criteria (Badri, 2019; Emmons and Vairaktarakis, 2012; Hsu et al., 2010; Li and Ierapetritou, 2007; Mahale, 2017; Ovacik and Uzsoy, 2012; Ramya and Chandrasekaran, 2013.)

An MIP model which has many aspects of our case (Unal et al., 2020) and mainly lag times between jobs. Due the shortage of data, about transportation lag times via cranes, we omitted this parameter however we can be compatible with this requirement on another phase. Another approach which is quite smart with a good performance is a decomposition method using mixed-integer and constraint programming (Harjunkoski and Grossmann, 2002). Constraint programming (CP) tend to perform very well in flow shop scheduling problems as it gives good feasible solutions in a short amount of time. Scheduling problems can naturally be decomposed into assignment and sequencing subproblems. So, the authors' strategy relies on either combining mixed-integer programming (MILP) to model the assignment part and constraint programming (CP) for modelling the sequencing part.
To the best of our knowledge the most relevant previous work appears in (Benda et al., 2019). The authors proposed an elegant methodology for solving large flow shop scheduling instances. The authors proposed a tree-based priority rule in terms of a well-performing decision tree (DT) for dispatching jobs. The proposed DT relies on high quality solutions, obtained using a constraint programming (CP) formulation. Novel aspects include a unified representation of job sequencing and machine assignment decisions, as well as the generation of random forests (RF) to face overfitting behaviour.

6.3 Model and/or Solution method (Demonstration)

The problem that we encounter is a *Flexible multistage flowshop problem with machine dependent setup times*. However, we have imposed different additional aspects in our model to imitate the real situation as accurate as possible. This means that there are some restrictions regarding the different products. For example, we do not allow a 'coil' product to be in a stage where bars are being processed. Another factor prohibits any job to go from one machine to another if these are part of the set of 'non-existing' paths. This feature reflects the fact that we cannot schedule a job to be processed between an automated and a manual machine.

6.3.1 Notation

At this section we present the basic notation that will be used in our optimization model. We note that each order has several different jobs that is required, namely every job in our case could be a specific product (i.e., a product with a Shape Code which may denote a bar with diameter Φ = 12 mm and 4 m length etc.). The factory receives the orders from the customers given a unique order ID to track the jobs that compose an order.

Sets

- $\mathcal{S}$: set of stages $(s = |S|)$
- $\mathcal{M}(s)$: set of machines in stage $s \in \mathcal{S}$, $(m(s) = |M(s)|)$
- $\mathcal{I}$: set of jobs
- $\hat{\mathcal{B}}$: set of forbidden (job,machine) combinations
- $\hat{\mathcal{M}}$: set of non-existing processing paths

Parameters

- T_i^d : due date of job $i \in \mathcal{I}$
- T_i^r : release date of job $i \in \mathcal{I}$
- T_m^s : setup time for machine $m \in \mathcal{M}(s)$
- T_{mi}^p : processing time in machine $m \in \mathcal{M}(s)$ for job $i \in \mathcal{I}$
- U : a positive big number, $U = \sum_m \sum_i T_{mi}^p$
- λ : weight coefficient between makespan-tardiness

Decision Variables

- y_{mi} : 1 iff on machine $m \in M(s)$ we schedule job $i \in I$ and 0 otherwise
- $x_{ii's}$: 1 iff we assign job i before job i' at stage $s \in S$ and 0 otherwise
- c_{is} : completion time of job $i \in I$ at stage $s \in S$
- C_{max} : makespan, the time that is required to finish the last job
- L_i : lateness of job $i \in I$, $L_i = c_{is} - T_i^d$
- T_i : 1 iff job $i \in I$ is tardy and 0 otherwise

6.3.2 Assumptions

After consultation with BRC people in charge we will create a deterministic model which captures the essential structure of Bay 3. With regards to maintenance, there are some historic data in paper format. Currently there is no periodic planning, and the maintenance is based on empirical rules. However, BRC will apply in the future a periodic maintenance plan based on the specifications of each different machine. So, at this phase we consider the maintenance as input, and we incorporate this aspect by a parameter providing whether the machine is available or not. The assumptions made for the development of the present MIP are as follows:

- All jobs are available at the start of time horizon.
- All jobs follow the same predefined order of stages.
- No preemption/ interruption is allowed.

- No job can be processed by more than one machine at the same time and no machine. can process more than one task at the same time (i.e., job slitting is not allowed).
- There should be no waiting time between consecutive job.
- Processing time is independent of the schedule.
- The machines are parallel unrelated which implies that the machines are not uniform and might have different processing and setup times for the same product.
- If a product is flagged as finished, then it cannot be processed again. So, reproduction is not allowed.

6.3.3 Mathematical Formulation

In this section we present our MILP for bay 3. We note that at every stage the factory can process only a specific set of jobs. There are three stages $s \in \{1,2,3\}$ and three different kind of jobs i $\in \{coil,\ cutting,\ bending\}$. We know in advance that the job 'coil' is processed in stage s=1, the job 'cutting' is processed in stage s=2 and finally the job 'bending' at stage s=3. To be consistent with the factory's production line we constructed the set $\hat{B}$ whose members are all the feasible combinations of jobs and machines.

$$minimize \quad \lambda C_{max} + (1-\lambda)\sum_{i\in I} T_i \tag{6.1}$$

$$s.t.$$

$$\sum_{m\in M(s)} Y_{mi} = 1 \qquad \forall i \in I, s \in S \tag{6.2}$$

$$C_{max} \geq c_{is} \qquad \forall i \in I, s \in S \tag{6.3}$$

$$c_{is} \geq \sum_{m\in M(s)} y_{mi}(T_i^r + T_m^s + T_{mi}^p) \qquad \forall i \in I, s \in S \tag{6.4}$$

$$c_{is} \leq c_{i,s+1} - \sum_{m\in M(s+1)} y_{mi}(T_{mi}^p + T_m^s) \qquad \forall i \in I, s < |S| \tag{6.5}$$

$$c_{i's} \geq c_{is} + T_{mi'}^p + T_m^s - U(3 - y_{mi} - y_{mi'} - x_{ii's}) \qquad \forall i,i' \in I, i < i', \forall s \in S, m \in M(s) \tag{6.6}$$

$$c_{is} \geq c_{i's} + T_{mi}^p + T_m^s - U(2 - y_{mi} - y_{mi'} + x_{ii's}) \qquad \forall i,i' \in I, i < i', \forall s \in S, m \in M(s) \tag{6.7}$$

$$y_{mi} = 0, \qquad \forall i \in I, m \in M, (i,m) \in \hat{B} \tag{6.8}$$

$$y_{mi} + y_{\bar{m}i} \leq 1 \qquad \forall i \in I, (m,\bar{m}) \in \hat{M} \tag{6.9}$$

$$L_i = \max\{c_{is} - T_i^d, 0\} \qquad \forall i \in I, s \in S \tag{6.10}$$

$$L_i \leq T_i U \qquad \forall i \in I \tag{6.11}$$

$$\sum_{i\in I} y_{mi}(T_{mi}^p + T_m^s) \leq \max_{i\in I}\{T_i^d - \sum_{s'\in S, s'>s} \min_{m'\in M(s'),\ (i,m')\notin \hat{B}} (T_{m'i}^p + T_{m'}^s)\} - \min_{i\in I}\{T_i^r + \sum_{s'\in S, s'<s} \min_{m'\in M(s'),(i,m')\notin \hat{B}} (T_{m'i}^p + T_{m'}^s)\}, \forall s \in S, m \in M(s) \tag{6.12}$$

$$x_{ii's}, y_{mi}, T_i \in \{0,1\},\ 0 \leq \lambda \leq 1, c_{is} \geq 0 \tag{6.13}$$

After discussions with BRC we established as criterion a convex sum of makespan and the number of tardy jobs, see (6.1).

We define Y_{mi} = 1 if job i is assigned on machine m. The set of constraints (6.2) ensure that every job is assigned to a machine at each stage, respecting the relationship connecting jobs to stages, as mentioned before the mathematical model. Constraints (6.3) link the makespan decision variable with the completion time of the last job to finish its processing. The next four sets of constraints (6.4) - (6.7) are related with the completion time of a job. More precisely constraint (6.4) refers that the completion time of a job i in a stage s should be at least the summation of release date, the processing time that the job needs to be done and the machines' setup time. The next constraint (6.5) guarantees the precedence sequence where each job cannot start its processing at stage s before it finishes at stage s-1. This set of constraints be active only for those jobs which need cutting and bending operations.

The next two set of constraints (6.6) - (6.7) prevent any two jobs from overlapping in a common machine. The difference of completion times between job i, which precedes job i should be at least the setup time plus the processing time of the first. From these two sets of constraints only one set will be active and the other will be redundant. At this point a small example could be helpful for the reader. First, we remind that the $x_{ii's}$ = 1 if job i precedes job i'. We can observe that we do not need the machine index m in the $x_{ii's}$ decision variables because the nature of the constraints and the relationship that exist between y_{mi} and $x_{ii's}$, hence these restrictions make sense only when we have jobs in a common machine. Let's assume that job job i precedes job i'on machine m at stage s so $y_{mi} = 1$, $x_{ii's} = 1$. If we substitute these values in the above constraints, we will take:

$$c_{i's} - T^{p}_{mi'} - T^{s}_{m} \geq c_{is}, \text{ so}$$

$$\text{starting time } i' \geq \text{finishing time } i \Rightarrow \text{True}$$

$$c_{is} - T^{p}_{mi} - T^{s}_{m} \geq c_{i's} - U \Rightarrow \text{trivial}$$

We can check that the first constraint implies that the starting time of job $i'(c_{i's} - T^{p}_{mi'} - T^{s}_{m})$ is at least the completion time of job i. However, the second constraint is redundant. So, the initial assumption which job i precedes job i' is hold.

Forbidden assignments are specified in (6.8), where $\hat{B}$ is a set of forbidden (job, machine) combinations. Using this constraint, we ensure that every job is going to be processed in the correct stage. Similarly, constraint (6.9) prohibits any job i to go from machine m to m' if these are part of the set of non-existing processing paths $\widehat{M}$.

Furthermore, constraint sets (6.10) - (6.11) referred to the tardiness of a job. In more detail, constraints (6.10) calculate the lateness of a job and specified only the positive lateness as tardiness $(L_i = \max\{c_{is} - T^{d}_{i}, 0\}$. Constraint (6.11) links the tardiness with the decision variables counting the number of tardy jobs, namely, if lateness is greater than zero $(L_i > 0)$, then the job is tardy T_i = 1. Finally, the next constraint will reduce the search space by adding some logical cutting plane. Constraint (6.12) reduces the search domain by making

sure that the total processing time of the jobs on a machine will fit between 1) the maximum due date subtracted with the shortest processing and setup times of all later stages, and 2) the minimum release date plus the shortest processing times of all earlier stages. Finally, constraints (6.13) ensure the integrallity constraints and the non-negativity as well.

6.4 Computational Experience

We now present a limited set of computational results from the application of our MIP on some randomly generated instances. Note that most computational studies in the literature are dominated by heuristic methodologies.

We performed all tests on a machine with Intel(R) Xeon(R) E5-2650 v2 2.6 GHz, 16 GB RAM, Windows 2007, using CPLEX solver. We want to emphasize on the statistics as the scale of the instances raise. A batch of test instances consist of 10 problems randomly generated. We fixed the number of machines as the production line of Bay 3 work with, and we investigate how the mathematical formulation reacts while we increase the number of jobs in relationship the objective goal. The percentage near the solution time is the proportion on how many problems were solved within the time limit condition which is 30 minutes in our case.

	makespan (sec)	tardy jobs (sec)	makespan-tardy (sec)
(n=10, m=9)	3.54 (100%)	0.35 (100%)	1.53 (100%)
(n=12, m=9)	2.57 (100%)	0.85 (100%)	171.45 (100%)
(n=14, m=9)	232.96 (70%)	1.16 (100%)	52.06 (60%)
(n=16, m=9)	435.48 (60%)	8.54 (100%)	323.4(50%)
(n=18, m=9)	446.53 (50%)	12.19 (100%)	1087,48(20%)
(n=20, m=9)	145.33 (40%)	68.59 (100%)	- (0%)

Table 11. Instances' solution times according to objective criterion

We can see every time we add two extra jobs the complexity is increased, and the solution time tends to grow and the percentage of solved problems within the time limits seems to deescalate. In addition, we observe that the combination of makespan-tardy jobs is the most computational expensive case having big solution times and small percentage of solvability.

7 Real-time Analytics

As already described in the deliverable D1.3, we define cognition as a complex process aiming to provide a proper understanding of the underlying industry process which starts with the detection of the variations in the process.

According to the process management theory, there are two types of process variations:

- Common cause variation – All processes have common cause variation. This variation is a normal part of any process. It demonstrates the true capability of a process.
- Special cause variation – This variation is not normal to the process. It is the result of exceptions in the process environment.

In the deliverable D3.1 we focused on the second type of variations, which are usually leading to the various types of anomalies.

In this deliverable, we focus on the first type of the variations, which can be considered as “normal” since they are based on the nature of the process. In the industry processes, one form of the special cause variations is related to the so-called phases in the processes. These are different operational modes (stages) which are based on the machine design.

For example, if a machine operates in three phases, it should be possible to detect the variations from the normal/usual/expected value, in the energy consumption in one of the phases (and not in general). This can be an early indicator of an anomalous behaviour of that phase and would require (re) optimisation,

These variations define the structure of a process and as such are important for the optimization of the process execution, as defined in our Cognitive Factory Model (reported in deliverable D3.1). Figure 9 depicts the difference between the general cognition model and the new one.

Briefly, in the new model the variations are better understood due to well defined context since the process structure is known (derived from data).

From the data analytics point of view, it is important to treat/process the phases separately to be able to properly understand the data and learn precise models. Otherwise, the learning process will produce models which are covering more than one process mode, without being able to generalize the knowledge about the process properly. Consequently, the services based on these models (e.g., anomaly detection) will not working precisely (e.g., generating too many false positive alarms).

From the cognition point of view, detecting this type of variations is of a high importance for the situation understanding, i.e., to decide if the reaction on the variations is requested. For example, by knowing that the machine is in the moving-head phase, there is an expectation that the usage of resources (e.g., energy) will be decreased, as well as the emission values should be reduced.

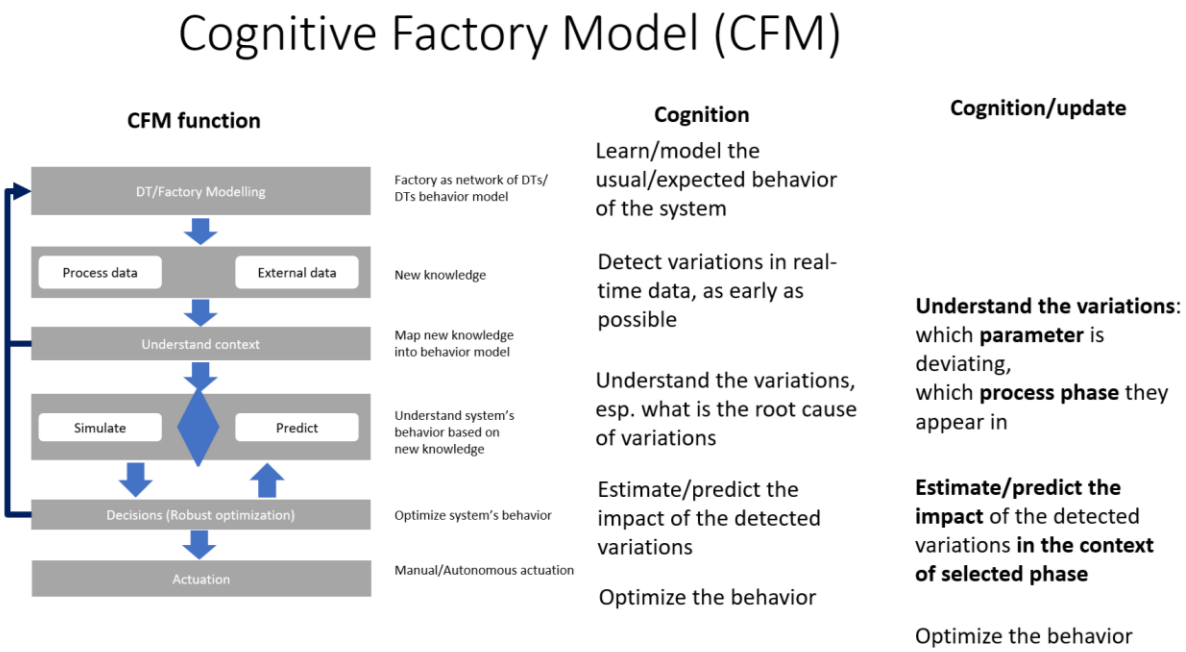

Figure 9. Updated cognition model

However, by considering that there can be various subphases in a process, an efficient detection of these variations is a difficult task. Indeed, there are two main approaches for the detection of phases:

- Model-driven, by knowing the physics of the process and selecting some of the process parameters for making the decision about the operating phase. For example, the parameter "depth" is important for drilling processes, for deciding if the process is in drilling/active or a passive phase.
- Data-driven, where the phases should be learned from the past data using some ML approaches.

The main constraint in the first method is that it requires a deep domain knowledge, whereas in some processes it can be missing.

The main advantage of the second method is that it can be applied to any process where the past data is available. The biggest challenge is to decide which data should be used in the data-driven process phases detection process. Usually, this should be the process data, as mentioned parameter "depth" in the drilling process.

In this deliverable we provide a novel approach for detecting process variations using energy consumption sensor. This sensor is a very common in the industry processes, since the energy consumption should be properly monitored in the context of various regulations, esp. from the environmental protection point of view. Moreover, it is planned to install additional energy sensors in at least two pilots (BRC, PIAZENCA), so that the relevance for this service is clear.

Since the data from the pilots is not available yet, we have used the energy consumption data available from other industry pilots. Since the format of the data obtained from different

energy sensors should be similar, we argue that the generalization of the proposed solution and its application on selected pilots is feasible.

The selected industry pilot is related to plasma cutting process, whereas the energy consumption sensor is attached to one of those machines. In the following text we present the results from that pilot.

7.1 Data-driven process (phase) variation detection

This section explores relevant details of designing, implementation and testing of plasma cutter detection system based on power sensors. The system is intended for detecting cutting phases of plasma cutter.

A solution for this system already exists, but is mostly based on processing video streams, from usual and special industry cameras. These cameras enable the clear visualization of the processes as if they were cold.

It is thought that system can be overburdened with a real-time video processing. Furthermore, expanses for using and maintaining such hardware are thought to be excessive for customers. The main goal of detection system is excluding camera sources completely or using them as an optional type of source if the system satisfies desired performances.

7.1.1 Requirements

To detect plasma cutter phases without the use of camera video sources, we can only rely on a power sensors data. The power sensor measures following parameters in three points (A, B. C):

-THDI- current harmonic distortion

-THDU- voltage harmonic distortion

Requirements include video sources from common cameras needed for labelling phases as cutting or not, as well as actual power sensors data pre-processed for neural network input.

7.1.2 Possible approaches

There are two main approaches for the phase detection:

- detection of cutting phase and
- classification every phase as cutting or not cutting.

Classification seems to be a good choice if we assume that we can detect the following phases:

- when machine does not work at all,
- when it works but does not move,
- when it moves but does not cut

and the last one is a phase we are looking for - when it cuts, according to the data we get from sensors.

Another open issue is whether we will need feature extraction. If we think about the nature of data (current, voltage), it seems like we could extract enough information from raw data for finding threshold values. Another approach is feature extraction for detecting patterns of every phase.

Regarding data preprocessing, an open issue is the segment to be detected. The question is if it is possible to detect cutting only from one row or we need to segment data. A row is a particular measurement from power sensor which is performed each 2-3 seconds and it is open if a particular phase can be detected according to a row or a window of continual rows (measurements) is needed, labeled as observed phase. According to the nature of data, it seems to be possible to detect cutting for every measurement (measurements are data from power/energy sensor).

7.1.3 Data preprocessing

This chapter focuses on approaches and constraints for dataset preprocessing.

7.1.3.1 First approach

Data from power sensors are labelled with corresponding phases using CPD (Change Point Detection) with a Binary segmentation algorithm in a background. In this context, change point is a particular point where the phase has been changed. In those points values for 'Active Power Total' parameter are abruptly increased or decreased. Though, we labelled all data between two change points as an appropriate phase and we got about 3000 measures per each phase. The illustration below shows the process of labelling using this method.

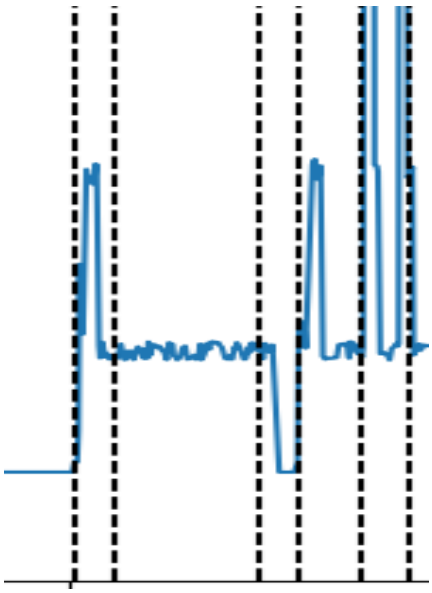

Figure 10. Labelling using CPD

There is one constraint which was discovered after labelling with CPD, and its limited usage of CPD for this use case. Plasma machine can use different power levels while performing

cutting on different materials. That means that some phases which are considered to be clearly cutting for one power level, would be active state period without cutting for higher power levels.

This phenomenon was the reason for changing the approach for data preprocessing. Better approach demands relying on a relation between the parameters, instead of the values, and that required some feature extraction from data we have.

7.1.3.2 *Second approach*

After observing the behaviour of parameters for different power levels and different phases, it was clear that correlations of some parameters stay the same in all observed cases, though it was used for data pre-processing. Correlation between all pairs of points where THDI is measured (a with b, b with c, and a with c) are extracted as new features. The illustration below shows the behaviour of parameters in different phases of cutting process.

The correlation is extracted on a window of 5 continual measurements, and after that rule-based logic is used for determining the class of the phase for each measurement. When this step is done, voting performed on set of window determines appropriate class for that window. Now, dataset is totally based on parameter relations, instead of values. Flow-chart below shows the steps for better understanding and the following illustration shows the rule-based logic for labelling.

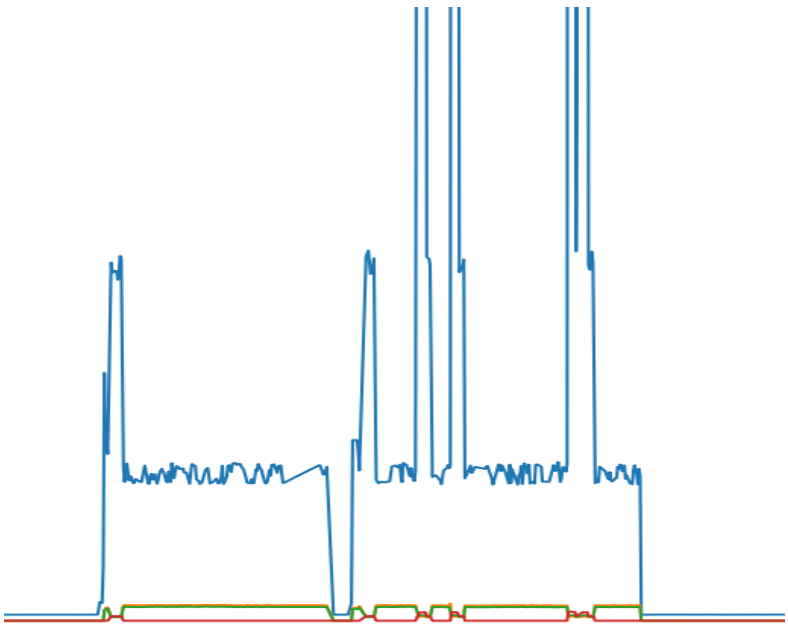

Figure 11. Behavior of parameters in different phases of cutting process

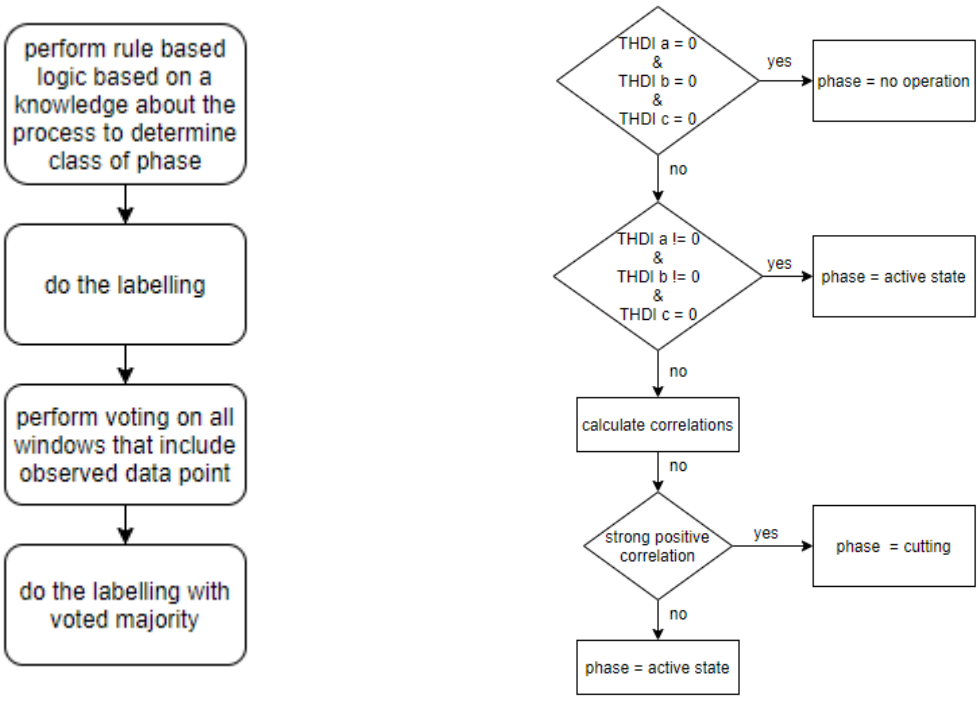

Figure 12. Steps and rule-based logic for labelling.

Furthermore, one more feature is added, it is a range for Active Power Total Average in observed window. Now, dataset is totally based on parameter relations, instead of values, so this idea is used for further processing. The main advantage of this is providing the possibility to detect phases of cutting process totally regardless of parameter values, because they can vary due to use of different power levels of plasma machine.

7.2 Classification methods

This chapter discusses different classification methods we used for phase detection, as well as results after performing each of them. We selected two methods KNN, Multinomial Logistic Regression.

7.2.1 KNN

KNN (k-nearest neighbours' algorithm) is commonly used for classification problems. In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbours, with the object being assigned to the class most common among its k nearest neighbours (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbour. Because of its simplicity it appears to be a good idea to start with.

7.2.2 Multinomial Logistic Regression

On their own, logistic regressions are only binary classifiers, meaning that they cannot handle target vectors with more than two classes. However, two clever extensions to logistic regression do just that. First, in one-vs-rest logistic regression (OVR) a separate model is trained for each class predicted whether an observation is that class or not (thus making it

a binary classification problem). It assumes that each classification problem (e.g., class 0 or not) is independent.

Alternatively, in multinomial logistic regression (MLR) the logistic function is replaced with a softmax function giving the probability of being the member of certain class.

Conclusion: KNN and MLR are common value-based classification methods, though their use for this case is questionable after discovering that we should not depend on the values.

7.3 Neural Networks

7.3.1 Fully Connected Neural Networks

Parameters

The neural network is built of 2 hidden layers with *tanh* activation function. *Softmax* is performed on an output layer, giving the probability of belonging to each class. *CategoricalCrossentropy* [2] is used as a loss function. *Adam*[3] is used as an optimizer.

7.3.2 Recurrent neural networks

Parameters

Recurrent neural network is built of three hidden layers. The first one is *LSTM* layer, second and third are Dense layers. *Softmax is* performed on an output layer, giving the probability of belonging to each class. *Categorical crossentropy* is used as a loss function. *Adam* is used as an optimizer.

In this section we will report results and give conclusions on usage of trained models for the purpose of detecting phases on plasma cutting machine relying on energy consumption sensors data.

In general, there are two main approaches:

- value dependent models (which use clear dataset from energy consumption sensors)
- value independent models (which use feature extraction from clear dataset)

All models can detect *cutting*, *active state* and *no operation* phase of plasma cutting machine.

Firstly, we will discuss value dependent models which are considerably simple solutions. Those are models with k-nearest neighbours' algorithm and multinomial logistic regression. They are very precise in detecting phases because they use parameter 'Active total power average' which can clearly separate those phases. On the other side, there is one constraint for using such models, and that is a fact claiming that levels of parameter 'Active total power average' can vary if plasma cutting machine uses different power levels in situation when it

[2] https://keras.io/api/losses/probabilistic_losses/#categoricalcrossentropy-class

[3] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

cut different materials. Conclusion is that we can use those models, which are extremely simple and fast, only if we know power levels that machine uses.

If that is not the case, there is a way to classify instances with appropriate phase, if we extract some features from row data. Parameters THDI (total harmonic distortion current) in point a, b and c are differently correlated in different phases and that can be used for classifying (look at illustration below). Models can be trained with fully connected and recurrent neural networks. Those models are value-independent but much more complex in comparison to *knn* and *mlr* models. Furthermore, there is one constraint that we are forced to use rolling window technique to perform calculation of correlations and other features on certain window. Window size is five points long, meaning that, we cannot calculate features for a particular point until we have data from whole window (delay of *win_size*/2, because the observed point is in the center of the window). Nevertheless, those models proved to be good at detecting phases but not as precise as value-based models.

7.4 Dataset for value-dependent models

Dataset for value-dependent models is just clear data from energy consumption sensors. Useful data from energy consumption sensor look like following:

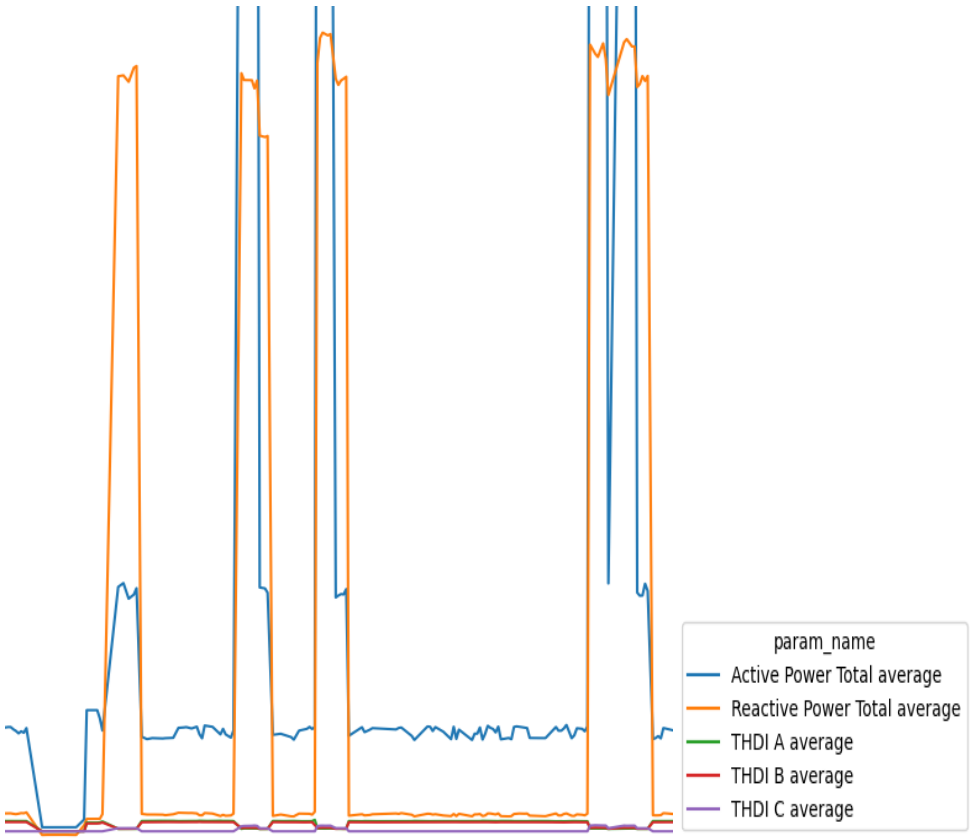

Figure 13. Visualization of data from energy consumption sensor

Sensors give more information like total harmonic distortion voltage in several points and temperature average, but those data are irrelevant for this use-case. Illustration above shows relevant data from energy consumption sensors that will be used in further processing.

7.5 Dataset for value-independent models

Extracted features are correlations between al pairs of THDI x parameters, min, max and mean of difference of all pairs of THDI x parameters, as well as range of active power total average in observed rolling window. Picture below shows dataset for value-independent models.

	corr_ac	corr_bc	corr_ab	active_power_range	diff_ac_mean	diff_ac_max	diff_ac_min	diff_ab_mean	diff_ab_max	diff_ab_min	diff_bc_mean	diff_bc_max	diff_bc_min
26751	-0.612372	-0.617914	0.936975	13537.7810	60.200000	104.10	-25.8	4.516667	11.4	-8.200	55.683333	92.80	-17.600
26752	-0.801784	-0.809040	0.936975	13537.7810	38.650000	103.70	-25.8	1.466667	11.4	-8.200	37.183333	92.30	-18.200
26753	-0.906327	-0.906327	1.000000	13465.6320	17.166667	102.50	-25.8	-1.600000	10.4	-8.200	18.766667	92.10	-18.200
26758	-0.345455	-0.090909	0.890909	11082.3500	-23.600000	-6.60	-29.4	-6.150000	-1.2	-7.400	-17.450000	-5.40	-22.000
26759	0.018182	0.385337	0.880771	11086.7490	-20.433333	-6.20	-29.4	-5.150000	-1.0	-7.400	-15.283333	-5.20	-22.000
26760	0.018182	0.385337	0.880771	11091.1480	-17.000000	-5.80	-29.4	-4.100000	-0.8	-7.400	-12.900000	-5.00	-22.000
26761	0.500000	0.709208	0.945611	11109.4650	-13.366667	-5.80	-29.4	-3.100000	-0.8	-7.400	-10.266667	-4.60	-22.000

Figure 14. Dataset for value-independent models

7.6 KNN and MLR training and results

Training dataset consists of clear data from energy consumption sensors (features are singled out in previous section). Training dataset is perfectly balanced and contains about 1500 instances per phase. Training execution time is negligibly small. Trained model is fastest and simplest possible and very precise at random checking as expected. *(Automatization of measuring precision process is not finished yet; only random checking can be performed right now. That is because we do not have accurate labels, and our idea was to use model that process video-frames and detect cutting phases to find intersect of results from all models).*

7.7 Fully connected neural network training and results

Neural network is built of 2 hidden layers with *tanh* activation function. *Softmax* is performed on an output layer, giving the probability of belonging to each class. *Categorical Crossentropy* is used as a loss function. *Adam* is used as an optimizer.

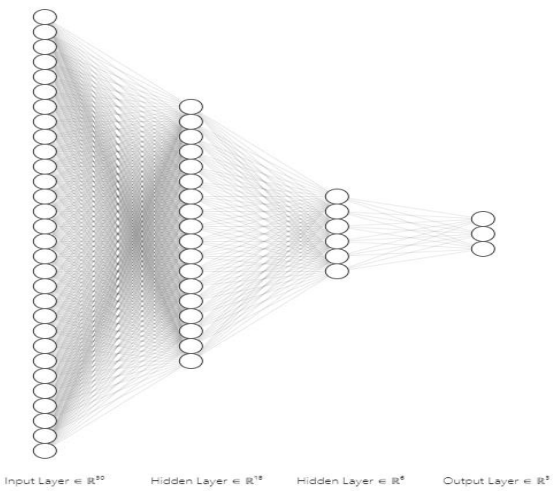

Figure 15. Layers of neural network

After 100 epochs of training, accuracy increased to 98%. Results are shown below.

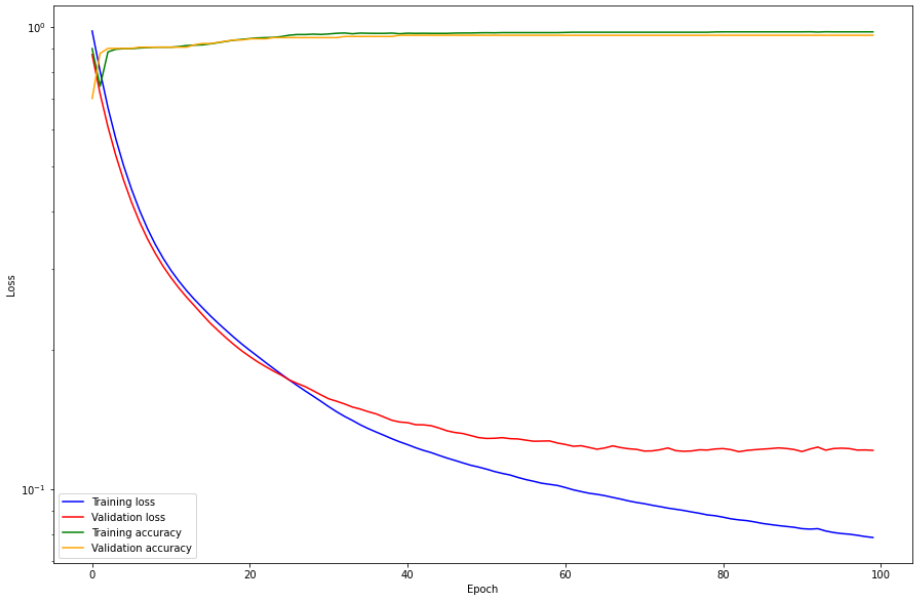

Figure 16. Results after training 100 epochs

7.8 Recurrent neural network training and results

Recurrent neural network is built of 3 hidden layers. The first one is *LSTM* layer, second and third are Dense layers. **Softmax** *is* performed on an output layer, giving the probability of belonging to each class. **Categorical Crossentropy** is used as a loss function. **Adam** is used as an optimizer.

Long short-term memory (**LSTM**) is an artificial recurrent neural (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural network, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data. LSTM networks are well-suited to classifying based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs.

Training accuracy is about 95%, as well as validation accuracy. Results are shown below. After 250 epochs training and validation loss become the smallest possible.

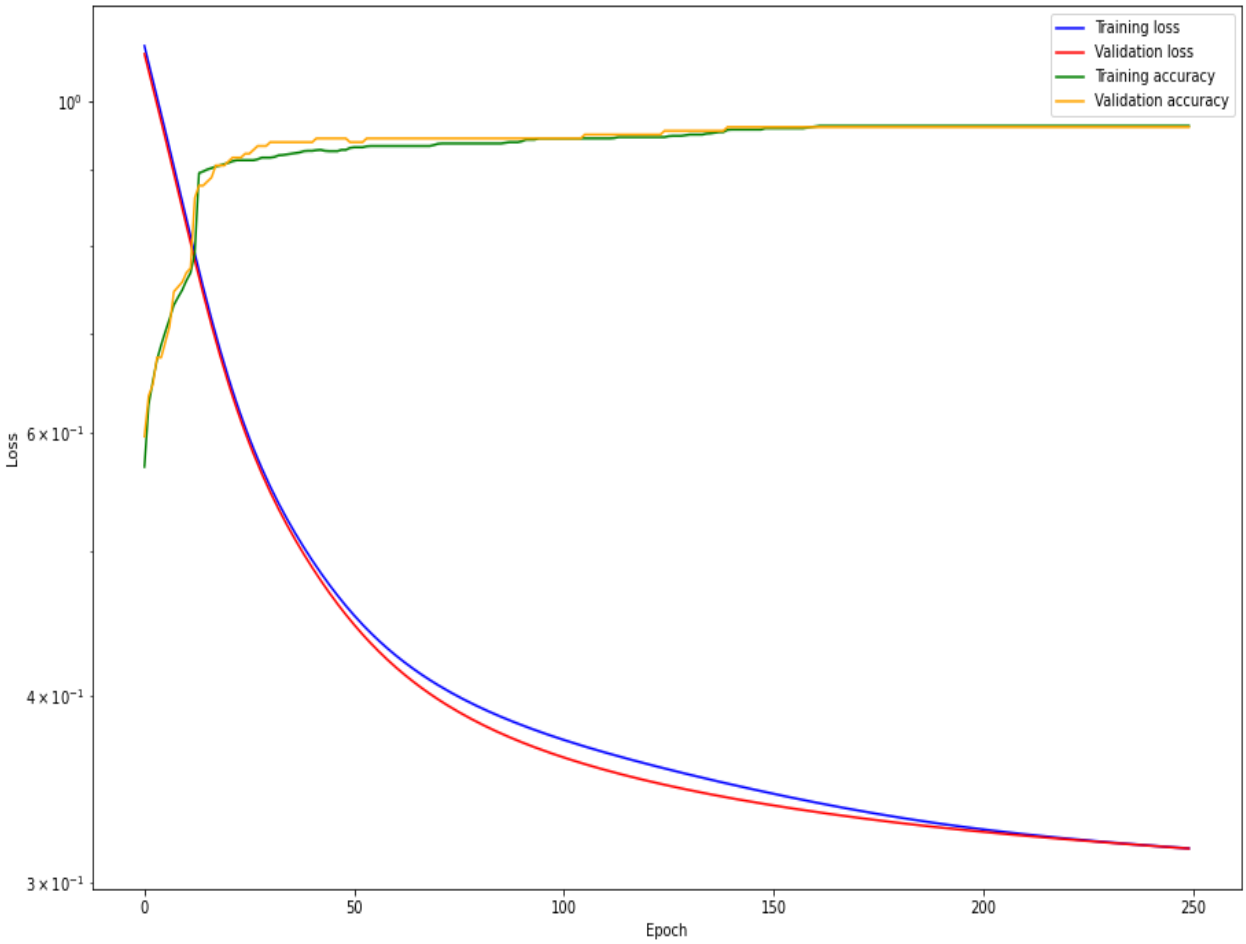

Figure 17. Results after training 250 epochs

References

[1] Albahri, T. A., Khor, C. S., Elsholkami, M., & Elkamel, A. (2018). Optimal design of petroleum refinery configuration using a model-based mixed-integer programming approach with practical approximation. Industrial & Engineering Chemistry Research, 57(22), 7555-7565.

[2] Allahverdi, A., Ng, C-T., Cheng, T. E. and Kovalyov, Y. A survey of scheduling problems with setup times or costs. EJOR, 187: 985-1032, 2008.

[3] Almeida Neto, E., Rodrigues, M.A., & Odloak, D. (2000). Robust predictive control of a gasoline debutanizer column. Brazilian Journal of Chemical Engineering, 17, 4-7.

[4] Andersen, P., & Petersen, N. C. (1993). A procedure for ranking efficient units in data envelopment analysis. Management science, 39(10), 1261-1264.

[5] Aschauer A., Roetzer F., Steinboeck A. and Kugi A. (2017). An Efficient Algorithm for Scheduling a Flexible Job Shop with Blocking and No-Wait Constraints. IFAC-PapersOnLine 50(1), 12490–12495.

[6] Aschauer A., Roetzer F., Steinboeck A. and Kugi A. (2018). Scheduling of a Flexible Job Shop with Multiple Constraints. IFAC-PapersOnLine 51(11), 1293–1298.

[7] Aspnes, Y., Azar, Y. Fiat, A., Plotkin, S., and Waarts, O. On-line Routing of Virtual Circuits with Applications to Load Balancing and Machine Scheduling. JACM 44(3):486–504, 1997.

[8] Azadeh A., Farahani M., Hosseinabadi Kalantari S.S. and Zarrin M. (2015) Solving a multi-objective open shop problem for multi-processors under preventive maintenance. International Journal of Advanced Manufacturing Technology

[9] Bajestani M., Banjevic A.D., Beck, J.C. (2014) Integrated maintenance planning and production scheduling with Markovian deteriorating machine conditions. International Journal of Production Research

[10] Barber, C. B., Dobkin, D. P., & Huhdanpaa, H. (1996). The quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software (TOMS), 22(4), 469-483.

[11] Badri, H. (2019). A parallel randomized approximation algorithm for single machine scheduling with applications to ow shop scheduling.

[12] Benda, F., Braune, R., Doerner, K. F., & Hartl, R. F. (2019). A machine learning approach for flow shop scheduling problems with alternative resources, sequence-dependent setup times, and blocking. OR Spectrum: Quantitative Approaches in Management, 41 (4), 871{893. https://doi.org/10.1007/s00291-019-00567-

[13] Belaid R., T'Kindt V., and Esswein C. (2012). Scheduling batches in flowshop with limited buffers in the shampoo industry. European Journal of Operational Research 223(2), 560–572.

[14] Ben Ali, M., Sassi, M., Gossa, M. and Harrath, Y. (2011) Simultaneous scheduling of production and maintenance tasks in the job shop. International Journal of Production Research

[15] Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., & Weglarz, J. (2014). Handbook on scheduling: From theory to applications. Springer Publishing Company, Incorporated.

[16] Branda A., Castellano D. Guizzi G, and Popolo V. (2020). Metaheuristics for the flow shop scheduling problem with maintenance activities integrated. Computers and Industrial Engineering

[17] Brucker P., Heitmann S., Hurink J., and Nieberg T. (2006). Job-shop scheduling with limited capacity buffers. OR Spectrum 28(2), 151–176.

[18] Brucker, P. Scheduling Algorithms. Springer, 1999

[19] Cui W.W, and Zhiqiang L. (2017) Minimizing the makespan on a single machine with flexible maintenances and jobs' release dates. Computers and Operations Research

[20] Charnes, A., Cooper, W. W., & Rhodes, E. (1978). Measuring the efficiency of decision making units. European journal of operational research, 2(6), 429-444.

[21] Correa, J., Verdugo, V., Verschae, J. Splitting versus setup trade-offs for scheduling to minimize weighted completion time. ORL, 44: 469-473, 2016.

[22] Doyle, J., & Green, R. (1994). Efficiency and cross-efficiency in DEA: Derivations, meanings and uses. Journal of the operational research society, 45(5), 567-578.

[23] de Gouvêa, M. T., & Odloak, D. (1998). One-layer real time optimization of LPG production in the FCC unit: procedure, advantages and disadvantages. Computers & Chemical Engineering, 22, S191-S198.

[24] Ehram S., Sadjadi S.J., Kamran S. (2010) Scheduling flow shops with condition-based maintenance constraint to minimize expected makespan. International Journal of Advanced Manufacturing Technology

[25] Eroglu, D. Y. and Ozmutlu, H. C. Solution method for a large-scale loom scheduling problem with machine eligibility and splitting property. TJTI, 108(12): 2154-2165, 2017.

[26] Emmons, I. H., & Vairaktarakis, G. (2012). Flow shop scheduling: Theoretical results, algorithms, and applications.

[27] Eroglu, D. Y., Ozmutlu, H. C. and Ozmutlu, S. Genetic algorithm with local search for the unrelated parallel machine scheduling problem with sequence-dependent setup times. IJPR, 52(19):5841-5856, 2014.

[28] Goel V., Slusky M., Van Hoeve W. J., Furman K.C., and Shao Y. (2015). Constraint programming for LNG ship scheduling and inventory management. European Journal of Operational Research 241(3), 662–673.

[29] Groflin H., Pham D.N., and Burgy R. (2011). The flexible blocking job shop with transfer and set-up times. Journal of Combinatorial Optimization 22(2), 121–144.

[30] Gholami M., Zandieh M., and Alem-Tabriz, A. (2009) Scheduling hybrid flow shop with sequence-dependent setup times and machines with random breakdowns. International Journal of Advanced Manufacturing Technology

[31] Han, Y., Geng, Z., Wang, Z., & Mu, P. (2016). Performance analysis and optimal temperature selection of ethylene cracking furnaces: a data envelopment analysis cross-model integrated analytic hierarchy process. Journal of analytical and applied pyrolysis, 122, 35-44.

[32] Harjunkoski, I., & Grossmann, I. E. (2002). Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. Comp. Chem. Engng, 26, 1533-1552.

[33] Hsu, C.-C., Huang, K.-C., & Wang, F.-J. (2010). Online scheduling of workfloow applications in grid environment. In P. Bellavista, R.-S. Chang, H.-C. Chao, S.-F. Lin, & P. M. A. Sloot (Eds.), Advances in grid and pervasive computing (pp. 300{310). Springer Berlin Heidelberg.

[34] Letsios, D., Bradley, J. T., Suraj, G., Misener, R. and Page, N. Approximate and robust bounded job start scheduling for Royal Mail delivery offices. JOS, 1-22, 2021.

[35] Lee, J-H., HoonJang, H. and Kim, H-J. Iterative job splitting algorithms for parallel machine scheduling with job splitting and setup resource constraints. JORS, 2020.

[36] Iyer, R. R., & Grossmann, I. E. (1997). Optimal multiperiod operational planning for utility systems. Computers & chemical engineering, 21(8), 787-800.

[37] Kemaloğlu, S., özgen Kuzu, E., & Gökçe, D. (2009). Model predictive control of a crude distillation unit an industrial application. IFAC Proceedings Volumes, 42(11), 880-885.

[38] Khor, C. S., Yeoh, X. Q., & Shah, N. (2011). Optimal design of petroleum refinery topology using a discrete optimization approach with logical constraints. Journal of Applied Sciences, 11(21), 3571-3578.

[39] Komaki G.M., Shaya S. and Malakooti B. (2018). Flow shop scheduling problems with assembly operations: a review and new trends. International Journal of Production Research 57(2), 2926-2955.

[40] Kuo, T. H., & Chang, C. T. (2008). Application of a mathematic programming model for integrated planning and scheduling of petroleum supply networks. Industrial & engineering chemistry research, 47(6), 1935-1954.

[41] Laborie P., Rogerie J., Shaw P., and Vilım P. (2018). IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG. Constraints 23(2), 210–250.

[42] Li, W., Hui, C. W., & Li, A. (2005). Integrating CDU, FCC and product blending models into refinery planning. Computers & chemical engineering, 29(9), 2010-2028.

[43] Li, Z., & Ierapetritou, M. (2007). Process scheduling under uncertainty: Review and challenges. Computers and Chemical Engineering, 32, 715-727.

[44] Liang, L., Wu, J., Cook, W. D., & Zhu, J. (2008). The DEA game cross-efficiency model and its Nash equilibrium. Operations research, 56(5), 1278-1288.

[45] Logendran R. and Talkington D. (1997) Analysis of cellular and functional manufacturing systems in the presence of machine breakdown International Journal of Production Economics

[46] Mahale, S. (2017). Developing a real time online scheduling system for a manufacturing service company: Achieving visibility (PhD Thesis). Lamar University.

[47] Mascis A. and Pacciarelli D. (2002). Job-shop scheduling with blocking and no-wait constraints. European Journal of Operational Research 143(3), 498–517.

[48] Mete, E., & Turkay, M. (2018). Energy network optimization in an oil refinery. In Computer Aided Chemical Engineering (Vol. 44, pp. 1897-1902).

[49] Mokhtari H. and Mehrdad D. (2015). Scheduling optimization of a stochastic flexible job-shop system with time-varying machine failure rate. Computers and Operations Research

[50] Moradi E., Ghomi F., and Zandieh M. (2010) An efficient architecture for scheduling flexible job-shop with machine availability constraints International Journal of Advanced Manufacturing Technology

[51] Moro, L. F., & Pinto, J. M. (2004). Mixed-integer programming approach for short-term crude oil scheduling. Industrial & engineering chemistry research, 43(1), 85-94.

[52] Ovacik, I., & Uzsoy, R. (2012). Decomposition methods for complex factory scheduling problems. Spinger Science & Business Media.

[53] Perez-Gonzalez P., Fernandez-Viagas V. and Framinan J.M. (2020) Permutation flowshop scheduling with periodic maintenance and makespan objective. Computers and Industrial Engineering

[54] Peyro,L.F. Models and an exact method for the Unrelated Parallel Machinescheduling problem with setups and resources. ESWA, 2020.

[55] Peyro, L.F., Ruiz, R. and Perea, F. Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times. COR, 81: 173-182, 2019

[56] Pimentel, C., Alvelos, F., Duarte, A. and Carvalho, J. Exact and heuristic approaches for lot splitting and scheduling on identical parallel machine. IJMTM, 22(1): 39-57, 2011.

[57] Pinto, J. M., & Moro, L. F. (2000). A mixed integer model for LPG scheduling. In Computer Aided Chemical Engineering (Vol. 8, pp. 1141-1146).

[58] Rasmussen K.M., Ejlertsen L.S.M., Pour S., Burke E.K., and Drake J.H. (2017). A hybrid Constraint Programming / Mixed Integer Programming framework for the preventive signaling maintenance crew scheduling problem. European Journal of Operational Research 269(1), 341–352.

[59] Rajkumar M., Asokan P. and Vamsikrishna V. (2010). A GRASP algorithm for flexible job-shop scheduling with maintenance constraints. International Journal of Production Research

[60] Rahmati S., Habib A., Ahmadi A. and Karimi B. (2018). Multi-objective evolutionary simulation based optimization mechanism for a novel stochastic reliability centered maintenance problem. Swarm and Evolutionary Computation

[61] Ramya, G., & Chandrasekaran, M. (2013). Solving job shop scheduling problem based on employee availability constraint. Materials and Diverse Technologies in Industry and Manufacture, 376, 197-206. https://doi.org/10.4028/www.scienti c.net/AMM.376.197

[62] Ruiz-Torres A.J., Paletta G., and Rym M.H. (2017). Makespan minimisation with sequence-dependent machine deterioration and maintenance events. International Journal of Production Research

[63] Roberti, R., and Toth P. Models and algorithms for the Asymmetric Traveling Salesman Problem: an experimental comparison. EJTL, 1:113–133, 2012.

[64] Rosales, O.A., Bello, F. A. and Alvarez, A. Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. IJAMT, 76:1705–1718, 2015.

[65] Sales, L. D. P. A., Luna, F. M. T. D., & Prata, B. D. A. (2018). An integrated optimization and simulation model for refinery planning including external loads and product evaluation. Brazilian Journal of Chemical Engineering, 35(1), 199-215.

[66] Serafini, P. Scheduling jobs on several machines with job splitting property. INFORMS J. Comp., 44:531-659 1996.

[67] Steiger, C.,Walder, H., Platzner, M., & Thiele, L. (2003). Online scheduling and placement of real-time tasks to partially reconfigurable devices. In: Proceedings of the 24th International Real-Time Systems Symposium, Cancun, 224-235.

[68] Trabelsi W., Sauvey C., and Sauer N. (2012). Heuristics and metaheuristics for mixed blocking constraints flowshop scheduling problems. Computers Operations Research 39(11), 2520–2527.

[69] Unal, A. T., A_gral_, S., & Ta_sk_n, Z. C. (2020). A strong integer programming formulation for hybrid flowshop scheduling. Journal of the Operational Research Society, 71 (12), 2042-2052. https://doi.org/10.1080/01605682.2019.165

[70] Unsal O. and Oguz C. (2013). Constraint programming approach to quay crane scheduling problem. Transportation Research Part E 59, 108–122.

[71] Vasconcelos, C. J., Maciel Filho, R., Spandri, R., & Wolf-Maciel, M. R. (2005). On-line optimization applied to large scale plants. In Computer Aided Chemical Engineering (Vol. 20, pp. 199-204).

[72] Wang S. and Yu J. (2010) An effective heuristic for flexible job-shop scheduling problem with maintenance activities. Computers and Industrial Engineering

[73] Wang S. and Liu M. (2014) Two-stage hybrid flow shop scheduling with preventive maintenance using multi-objective tabu search method. International Journal of Production Research

[74] Wang, J-B. and Wang, J-J. Research on scheduling with job-dependent learning effect and convex resource-dependent processing times. IJPR, 53 (19): 5826-5836, 2015.

[75] Xu S., Dong W., Jin M. and Wang L. (2020). Single-machine scheduling with fixed or flexible maintenance. Computers and Industrial Engineering

[76] Yaurima V., Burtseva L., and Tchernykh A. (2009). Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers. Computers and Industrial Engineering 56(4), 1452–1463.

[77] Yu T., Sun H. and Jun H. (2021) Scheduling proportionate flow shops with preventive machine maintenance. International Journal of Production Economics

[78] Zandieh M., Khatami A.R., Rahmati Seyed., Habib A. (2017) Flexible job shop scheduling under condition-based maintenance: Improved version of imperialist competitive algorithm. Applied Soft Computing Journal

[79] Zandieh, M. Fatemi Ghomi, S. M.T. (2009) Scheduling sequence-dependent setup time job shops with preventive maintenance Naderi, International Journal of Advanced Manufacturing Technology

[80] Zanin, A. C., de Gouvea, M. T., & Odloak, D. (2000). Industrial implementation of a real-time optimization strategy for maximizing production of LPG in a FCC unit. Computers & Chemical Engineering, 24(2-7), 525-531.

[81] Zanin, A. C., De Gouvea, M. T., & Odloak, D. (2002). Integrating real-time optimization into the model predictive controller of the FCC system. Control Engineering Practice, 10(8), 819-831.

[82] Zheng Y., Lian L., and Mesghouni K. (2014) Comparative study of heuristics algorithms in solving flexible job shop scheduling problem with condition-based maintenance. Journal of Industrial Engineering and Management

Appendix 1 – TUPRAS Input and Output Data Classes

The input data provided to the optimisation module are grouped in the following public classes.

```
   public class ProcessInstance
{
    public List<InputFeed> InputFeeds { set; get; }
    public List<OutputTank> OutputTanks { set; get; }
    public Specifications Specs { set; get; }
    public OptSettings Settings { set; get; }
}

public class InputFeed
{
    public string InputNodeID { set; get; }
    public double IF_i { set; get; }
    public double ISU_i { set; get;
    public double IC2_i { set; get; }
    public double IC5_i { set; get; }
}

public class OutputTank
{
    public string OutputNodeID { set; get; }
    public double Q_total_i { set; get; }
    public double Q_start_i { set; get; }
    public double QC5_start_i { set; get; }
    public double QC2_start_i { set; get; }
    public double QSU_start_i { set; get; }
}

public class Specifications
{
    public double SU { set; get; }
    public double C2 { set; get; }
    public double C5 { set; get; }
    public double C2C5 { set; get; }
}


public class OptSettings
{
    public double Horizon { set; get; }
    public double TimeToOptimize { set; get; }
}

public class OperationalScenarios
{
    public List<UnitScenario> UnitScenarios { set; get; }
}

public class UnitScenario
{
    public string NodeID { set; get; }
    public string ScenarioID_i_s { set; get; }
    public double E_i_s { set; get; }
    public List<LinkScenario> LinkScenarios { set; get; }
```

```
}

public class LinkScenario
{
    public string LinkID { set; get; }
    public double CAP_ij { set; get; }
    public double PF_i_j_s { set; get; }
    public double PC5_i_j_s { set; get; }
    public double PC2_i_j_s { set; get; }
    public double PSU_i_j_s { set; get; }
}

public class Root
{
    public int Id { get; set; }
    public string Name { get; set; }
    public Definition Definition { get; set; }
}

public class Parameter
{
    public string Symbol { get; set; }
    public string Description { get; set; }
    public string Unit { get; set; }
    public double Value { get; set; }
}

public class Stock
{
    public double Quantity { get; set; }
    public double ExtraQuantity { get; set; }
    public int Consume { get; set; }
    public bool Enabled { get; set; }
}

public class Node
{
    [JsonProperty("$Type")]
    public string Type { get; set; }
    public int SpecificationMethod { get; set; }
    public string ScriptSource { get; set; }
    public List<Parameter> Parameters { get; set; }
    public string Stage { get; set; }
    public bool Solved { get; set; }
    public string Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public List<Stock> Stocks { get; set; }
    public bool? OneToOne { get; set; }
}

public class Flow
{
    public string Resource { get; set; }
    public string Name { get; set; }
    public double Quantity { get; set; }
    public bool Manual { get; set; }
    public bool Calculated { get; set; }
    public string Formula { get; set; }
```

```
    public double Factor { get; set; }
}

public class Link
{
    public string Source { get; set; }
    public string Target { get; set; }
    public List<Flow> Flows { get; set; }
    public string Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
}

public class Resource
{
    public string Unit { get; set; }
    public string Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
}

public class Definition
{
    public List<Node> Nodes { get; set; }
    public List<Link> Links { get; set; }
    public object Stages { get; set; }
    public List<Resource> Resources { get; set; }
    public List<Parameter> Parameters { get; set; }
    public double Tolerance { get; set; }
}
```

The output data provided by the optimisation module are grouped in the following public classes.

```
public class Solution
{
    public List<SelectedScenario> SolutionScenarios { set; get; }
    public double TotalEnergy { set; get; }
    public List<SpecsKPIs> OutputKPIs { set; get; }
    public SolutionKPIs SolKPIs { set; get; }
}

public class SelectedScenario
{
    public string NodeID { set; get; }
    public string ScenarioID { set; get; }
    public string OptID { set; get; }
}

public class SolutionKPIs
{
    public bool FoundSolution { set; get; }
    public double TimeToSolveMillisec { set; get; }
    public double TimeToInitializeMillisec { set; get; }
}

public class SpecsKPIs
{
    public string OutputNodeID { set; get; }
```

```
        public double Quantity { set; get; }
        public double SUperc { set; get; }
        public double C2perc { set; get; }
        public double C5perc { set; get; }
        public double C2C5perc { set; get; }
    }
```

Appendix 2 – PIACENZA Input and Output Data Classes

The input data provided to the optimisation module are grouped in the following public classes.

```
public class GeneralInfo
   {
      public string startDate { get; set; }
      public string endDate { get; set; }
     public int currentTotalSetupTime {get; set; }
   }

 public class Order
   {
      public string status { get; set; }
      public int chainID { get; set; }
      public int partID { get; set; }.
      public int loomID { get; set; }
      public double targetMeters { get; set; }
      public double kStrokes { get; set; }
      public string deliveryDate { get; set; }
      public string type { get; set; }
      public int fabricType { get; set; }
      public int ca { get; set; }
      public int cc { get; set; }
      public int strokesPerMeter { get; set; }
      public int yarns { get; set; }
      public int drawing { get; set; }
      public int variant { get; set; }
      public int incom { get; set; }
      public int comb { get; set; }
      public int combHeight { get; set; }
   }

public class Loom
   {
      public int loomID { get; set; }
      public int loomSpeed { get; set; }     /
   }

public class Worker

   {
      public int workGroups { get; set; }
      public string startDatetime { get; set; }
      public string endDatetime { get; set; }
   }

public class Root
   {
      public GeneralInfo general_info { get; set; }
      public List<Order> orders { get; set; }
      public List<Loom> looms { get; set; }
      public List<Worker> workers { get; set; }
   }
```

The output data provided by the optimisation module are grouped in the following public classes.

```
public class Order
   {
      public int chainID { get; set; }
      public int partID { get; set; }
      public string type { get; set; }
      public string deliveryDate { get; set; },
      public int loomID { get; set; }.
      public int setupTime { get; set; }
      public string setupStartTime { get; set; }
      public string setupEndTime { get; set; }
      public string processStartTime { get; set; }
      public string processEndTime { get; set; }
      public int processingTime { get; set; }.
      public int targetMeters { get; set; }
      public double tardiness { get; set;
   }

   public class LoomsOrderSequence
   {
      public string _loomID { get; set; }
   }

   public class ObjectiveValues
   {
      public int makespan { get; set; }
      public double totalTardiness { get; set; }
  }

 public class Root
   {
      public List<Order> orders { get; set; }
      public LoomsOrderSequence loomsOrderSequence { get; set; }
      public ObjectiveValues objectiveValues { get; set; }
   }
```

Appendix 3 – CONTINENTAL Input and Output Data Classes

Class	**Workstation**	
	Name	*Type*
	WorkplaceID [PK]	Int
	WorkplaceTypeID [FK]	Int
	ProductionLineID [FK]	Int
	SequenceInLine	Int
	WorkplaceName	String

Class	**WorkplaceTypes**	
	Name	*Type*
	WorkplaceTypeID	Int
	WorkplaceTypeName	String

Class	**ProductionLineTypes**	
	Name	*Type*
	ProductionLineTypeID	Int
	ProductionLineType	String

Class	**ProductionLines**	
	Name	*Type*
	ProductionLineID	Int
	ProductionLineTypeID	Int
	ProductionLineName	String

Class	**Storage_Zones**	
	Name	*Type*
	StorageZoneID [PK]	Int
	StorageZoneName	String
	StorageZoneCapacity	Int

Class	**Product_Inventory**	
	Name	*Type*
	ProductID	Int
	ProductQuantity	Int
	StorageZoneID	Int

Class	**Resource_Inventory**	
	Name	*Type*
	ResourceID	Int
	Quantity	Int

Class:	**Product_Family**	
	Name	*Type*
	ProductFamilyID [PK]	Int
	ProductFamilyName	String
	ProductFamilyDescription	String

Class	**Product**	
	Name	*Type*
	ProductID	Int
	ProductName	String
	ProductFamilyID	Int
	SourceLineTypeID	Int
	EndLineTypeID	Int
	ProductDescription	String

Class	**Product_Family_Setup_Times**	
	Name	*Type*
	ProductFamilyID	Int
	ProductionLineTypeID	Int
	SetupTime [sec]	Double

Class	**Production_Orders**	
	Name	*Type*
	OrderID	Int
	OrderName	String
	ProductID	Int
	Quantity	Int
	MaxQuantity	Int
	ReleaseDate	Datetime
	Priority	Int
	DueDate	Datetime

Class	**Product_Processing_Times**	
	Name	*Type*
	ProductID	Int
	WorkplaceID	Int
	IdealProcessingTime [sec]	Double
	RealProcessingTime [sec]	Double

Class	**Machine_Maintenance_Activities**	
	WorkplaceID	Int
	MaintenanceStartTime	Double
	MaintenanceEndTime	Double
	MaintenanceDurationTime	Double

Class	**Production BOM**	
	WorkplaceID	Int
	RequiredProductID	Int
	Multiplicity	Double
	ProductID	Int

Class	**Resource BOM**	
	WorkplaceID	Int
	ResourceID	Int
	Multiplicity	Double
	ProductID	Int

Class	**Resources**	
	ResourceID	Int
	ResourceName	String
	ResourceDescription	String

Class	**Production Schedule**	
	JobID	Int
	OrderID	Int
	Quantity	Int
	LineID	Int
	SequenceID	Int

Class	**Scheduled Maintenance Activities**	
	WorkstationID	Int
	MaintenanceStart	DateTime
	MaintenanceEnd	DateTime
	MaintenanceDuration	Int

Class	**UnScheduled Maintenance Activities**	
	WorkstationID	Int
	MaintenanceStart	DateTime
	MaintenanceEnd	DateTime
	MaintenanceDuration	Int

Appendix 4 – BRC Input and Output Data Classes

The input data provided to the optimisation module are grouped in the following public classes.

```
public class GeneralInfo
 {
        public string startDate { get; set; }
        public string endDate { get; set; }
        public int currentTotalSetupTime {get; set; }
        public float lambda { get; set; }
  }


public class InputOrder
 {
        public int orderID { get; set; }
        public string deliveryDate { get; set; }
 }

public class InputJob
{
        public int jobID { get; set; }
        public int parentID { get; set; }
        public string deliveryDate { get; set; }
        public string jobType { get; set; }
}

public class InputMachine
{
        public int machineID { get; set; }
        public string machineStatus { get; set; }
        public string machineType { get; set; }
}


public class ProcessingAndSetupTimes
{
        public int machineID { get; set; }
        public int jobID { get; set; }
        public float setupTime { get; set; }
        public float processingTime { get; set; }
}
```

The output data provided by the optimisation module are grouped in the following public classes.

```
public class OutputOrder
{
        public int orderID { get; set; }
        public int tardyJobs { get; set; }
}

public class OuputJob
{
        public int jobID { get; set; }
        public int parentID { get; set; }
```

```
            public List<Machine> machineID { get; set; }
            public List<string> startTime { get; set; }
            public List<string> completionTime { get; set; }
}
public class OutputMachine
{
    public int machineID { get; set; }
    public List<int> jobID { get; set; }
}

public class ObjectiveValues
{
        public int makespan { get; set; }
        public int totalTardiness { get; set; }
        public double totalLateness { get; set; }
}
```

Appendix 5 – Analytics Classes

```
z public class Datapoint
{
        private string timestamp;
        private float value;
        public string timestampGetSet {get; set;};
        public string valueGetSet {get; set;};
}

public class AnalysisValue
{
        private string analysisName;
        private string timestamp;
        private float value;
        private bool outlier;
        public string analysisNameGetSet {get; set;};
        public string timestampGetSet {get; set;};
        public float valueGetSet {get; set;};
        public bool outlierGetSet {get; set;};
}

public class Data
{
        private string mongoId;
        private string instanceId;
        private string arrivedTimestamp;
        private string type;
        private string startTimestamp;
        private string endTimestamp;
        private Dictionary <string, List<Datapoint>> parameterValues;
        public string mongoIdGetSet {get; set;};
        public string instanceIdGetSet {get; set;};
        public string arrivedTimestampGetSet {get; set;};
        public string typeGetSet {get; set;};
        public string startTimestampGetSet {get; set;};
        public string endTimestampGetSet {get; set;};
        public Dictionary parameterValuesGetSet {get; set;};
}

public class ModelDataResult
{
        private string mongoId;
        private string instanceId;
        private string arrivedTimestamp;
        private string dataId;
        private Dictionary <string, List<AnalysisValue>> analysisValues;
        private Dictionary<string, List<string>> outlierDatapoints;
        public string mongoIdGetSet {get; set;};
        public string instanceIdGetSet {get; set;};
        public string arrivedTimestampGetSet {get; set;};
        public string dataIdGetSet {get; set;};
        public Dictionary analysisValuesGetSet {get; set;};
        public Dictionary outlierDatapointsGetSet {get; set;};
}

public class DataProvider
{
```

```
        private string ipAddress;
        private int port;
        private string databaseName;
        private string username;
        private string password;
        public string ipAddressGetSet {get; set;};
        public int portGetSet {get; set;};
        public string databaseNameGetSet {get; set;};
        public string usernameGetSet {get; set;};
        public string passwordGetSet {get; set;};
        public bool writeData(Data object);
        public Data readData(string instanceId);
        public bool writeModelDataResult(ModelDataResult object);
        public Data readModelDataResult(string instanceId);
}

public class FTPReader
{
        private string ipAddress;
        private int port;
        private string username;
        private string password;
        public string ipAddressGetSet {get; set;};
        public int portGetSet {get; set;};
        public string usernameGetSet {get; set;};
        public string passwordGetSet {get; set;};
        public bool readDataFromServer(string instanceId);
}

public class Executor
{
        private DataProvider dp;
        private FTPReader ftp;
        public DataProvider ipAddressGetSet {get; set;};
        public FTPReader portGetSet {get; set;};
        public List<AnalysisValue> perform_knn(Data object);
        public List<AnalysisValue> perform_mlr(Data object);
        public List<AnalysisValue> perform_nn(Data object);
        public List<AnalysisValue> perform_rnn(Data object);
        public List<AnalysisValue> perform_video_processing(Data object);
        public ModelDataResult generate_MDR(List<List<AnalysisValue>>);
}
```