FACTLOG
www.factlog.eu

ENERGY-AWARE FACTORY ANALYTICS FOR PROCESS INDUSTRIES

Deliverable D5.2

Robust and energy-aware planning and scheduling

Version 1.0	**Lead Partner** AUEB
Date 31/03/2022	**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

This document reflects the opinion of the authors only.

While the information contained herein is believed to be accurate, neither the FACTLOG consortium as a whole, nor any of its members, their officers, employees or agents make no warranty that this material is capable of use, or that use of the information is free from risk and accept no liability for loss or damage suffered by any person in respect of any inaccuracy or omission.

Executive Summary

This deliverable reports on the progress conducted in WP5 Robust Optimization Methods relating to the implementation of the State-of-the-art Optimization Methods of the FACTLOG project. It reflects the work performed in the context of the project Task 5.2 Robust Energy-aware production scheduling and T5.3 Resource-aware production planning, and the outcomes thereof. Following the implementation of the overall FACTLOG system and the overall project evolution in the current tasks the progressive evolution of the algorithms and methods as well as their implementation (in line with T5.4 Robust Optimization as a Service) took place and as such, this deliverable proceeds with describing the updates on the optimization methods as well as the updated version of the optimization toolkit implementation.

Starting from the toolkit itself, as it was designed to be modular, expandable, and capable to be adapted and introduced in different cases, it has progressed to be able to handle all pilots' incoming data from the FACTLOG infrastructure and the pilots themselves. It solves the problems presented in D5.1 and follows the respective design and use cases. Optimization is initiated on an ad hoc basis with respective received signals from the FACTLOG ecosystem and output is return respectively.

Besides the actual development of the algorithms for the optimization engine, progress was made with respect to the proposed optimization approaches, algorithms and tools in the corresponding cases. Starting from the TUPRAS case relevant to the Oil Refineries and on-specs Liquefied petroleum gas (LPG) production, a mathematical programming approach to minimize the energy consumed for planning on-specs LPG production was developed and integrated in the Optimization toolkit. In this case, the problem is formulated as a Mixed Integer Linear Program (MILP) that integrates network flow and blending constraints in order to identify the most energy-efficient combination of configurations for all process units of the LPG purification process to achieve on-specs LPG production by calculating the optimal on-specs recovery plan. The MILP presented in D5.1 is further updated in this deliverable leading to a crisper presentation of the LPG purification process. Additional work includes the examination of the application of Data Envelopment Analysis assessment for identifying the dominant operational scenarios and hence reducing the solution space, the exploration of approaches such as Chance Constrained Programming and Interval Linear Programming to address uncertainty in the level of impurities in the input feed as well as handling the input feed rate and their incorporation in the proposed MILP approach. Lastly, the presentation of the concept of state-aware optimization which also utilizes other technological services of FACTLOG, such as the corresponding Machine Learning and Simulation tools.

Moving forward from TUPRAS to the second pilot and respective problem solved by the optimization module we have the PIACENZA case. In the PIACENZA case, there is a three-fold progress: (a) the Weaving scheduling problem (b) solving the (Parallel Machine Scheduling) PMS problem on unrelated machines with sequence-dependent setup times, job splitting and resource constraints and (c) design of effective exact methods. Therefore, in this case we provide novel and effective lower bounds and a three-stage heuristic for the makespan minimization problem identified at the pilot case. Additionally, we have numerically evaluated the algorithms developed on benchmark instances, as well as on experiments based on real datasets. Additional experiments enabled us to provide important

findings on the problem parameters as business consultation thus deriving to policies for unexpected events

The third 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 and the large-scale size of the problem. That combined with the inherent difficulty in needing cranes to load / unload machines create major bottlenecks in the production process. However, a significant progress was the incorporation of the cranes' movements and imitation of the process as accurate as possible. The most challenging issue faced was the tracking of the starting and the ending point for each job which needed to be moved. The goal of optimization in this case was to find the optimal production schedule in relation to the makespan or the total lateness of tardy jobs. The BRC case was solved utilizing Mixed-Integer Linear Programming (MILP) for an extended flexible multistage flowshop problem with machine dependent setup times. Nevertheless, to capture the cranes' movement the MILP is transformed to a Mixed Integer Quadratic Problem but in future research we can linearize it. 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 lateness. Finally, by finding some Lower Bound (LB) for the decision variables we obtained better computational times.

Lastly, the fourth 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. A rigorous Constraint Programming formulation is proposed for modeling and solving the problem. Results on synthetic and real data sets validate the applicability of the model and demonstrate the efficiency, effectiveness and scalability of the proposed Constrained Programming (CP) approach.

Revision History

Revision	Date	Description	Organisation
0.1	1/03/2022	ToC	AUEB
0.2	10/03/2022	Provision of Optimization Input for the cases of PIA, TUPRAS, BRC	AUEB, UNIPI
0.3	15/03/2022	Released for Internal review	AUEB
0.4	22/03/2022	Addition of CONT case and minor revisions	AUEB
0.5	27/03/2022	Provision of Review comments	TUC, QLECTOR
1.0	30/03/2022	Introduction of review comments and finalized document for upload	AUEB

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
UNIPI	Stathis Plitsos	stathisp@aueb.gr
AUEB	Kyriakos Bitsis	bad19024@uom.edu.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

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Purpose and Scope

WP5 is responsible for the provision of the optimization for the FACTLOG project. To this end, T5.2 Robust Energy-Aware production scheduling and T5.3 Resource-aware production planning, develop and provide resource-aware algorithms for advanced scheduling of production that support decision-making in the context of the pilots. In these tasks, the enhancement of the developed optimization algorithms takes place in all cases in order to further extend the capabilities of the Optimization engine taking under consideration textbook and state-of-the-art solution approaches. This deliverable thus reports on the former aspects reflecting the work performed in the two tasks and in the direction of the completion of the Optimization Toolkit.

1.2 Relation with other Deliverables

This deliverable is directly related with two deliverables. Initially it receives information from the D5.1 Real-time re-optimization which constitutes its predecessor having the initial examination of the cases as well as the initial approach of the MILPs. The second with a direct relationship is D5.3 FACTLOG optimization toolkit and service where the complete implementation of the overall toolkit will be presented. As this deliverable relates to the overall optimization in the FACTLOG project it also relates with deliverables as “D3.4 Proactive Cognitive Plants”, “D6.2 Data Collection Framework (Final Version)” and “D6.6 Integrated Package and Platform (Final Version)” and remaining deliverables and milestones feeding in for Optimization.

1.3 Structure of the Document

As this deliverable constitutes the evolution of the D5.1 Real-time re-optimization, it follows a similar structure where Section 2 presents the Optimization Module Update, and from then on, the sections document the work done on each pilot in terms of optimization. Specifically, it presents (a) additional background as the project and optimization work progresses, (b) the contribution and progress conducted, (c) updates on the problem, (d) updates of the solution approach, (e) evaluation – results and lastly the respective references. This presentation flow appears in Section 3 regarding the TUPRAS optimization, in Section 4 on the PIACENZA case, in Section 5 concerning the BRC case and in Section 6 presenting the CONTINENTAL case.

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 Application Programming Interface (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 a new operational model.
- Use Case 8 (UC8): Store a new operational scenario.
- Use Case 9 (UC9): Store the solution of a complete optimization job.

The image below illustrates the set of these functional requirements:

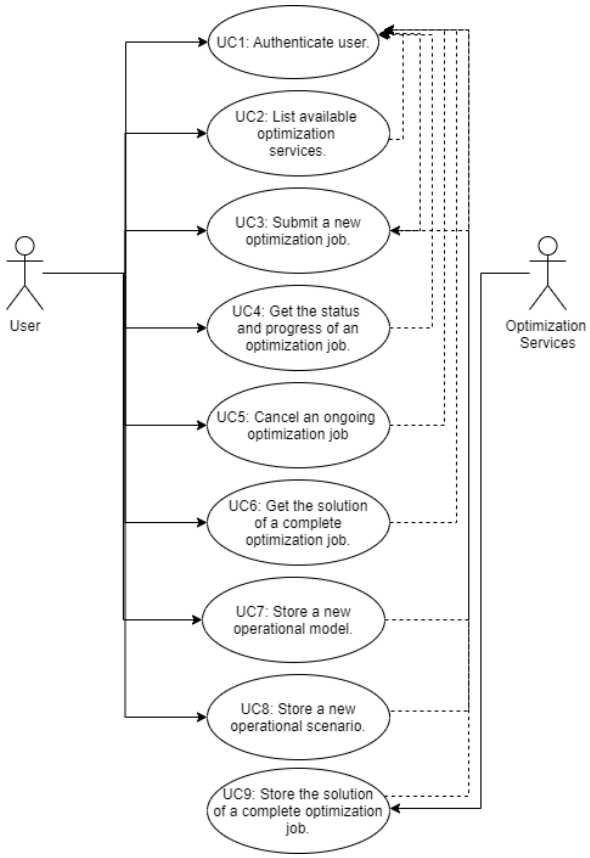

Figure 1. optEng Functional Requirements

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, operational_scenario_uuid, 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 completed.	
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 a new operational model.	
Brief description	This use case states the actions taken to store a new operational model.	
Primary Actors	User	
Pre-conditions		
Post-conditions	The operational model data is successfully stored.	
Basic flows	Tasks	Information required
	1. The User submits the operational model data.	operational_model
	2. The system stores the model.	Uuid, operational_model
Alternative flows	Tasks	Information required
	If an error occurs, the system returns the respective error code.	Error_code

Use Case 8	Store a new operational scenario.
Brief description	This use case states the actions taken to store a new operational scenario.

Primary Actors	User	
Pre-conditions	An operational model is already stored.	
Post-conditions	The operational scenario data is successfully stored.	
Basic flows	Tasks	Information required
	1. The User submits the operational scenario data.	operational_scenario
	2. The system stores the scenario.	Model_uuid, operational_scenario
Alternative flows	Tasks	Information required
	If an error occurs, the system returns the respective error code.	Error_code

Use Case 9	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 completed.	
Post-conditions	The solution data is successfully stored.	
Basic flows	Tasks	Information required
	3. The Optimization Services submit the uuid and the solution data.	Uuid, solution_data
	4. 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.

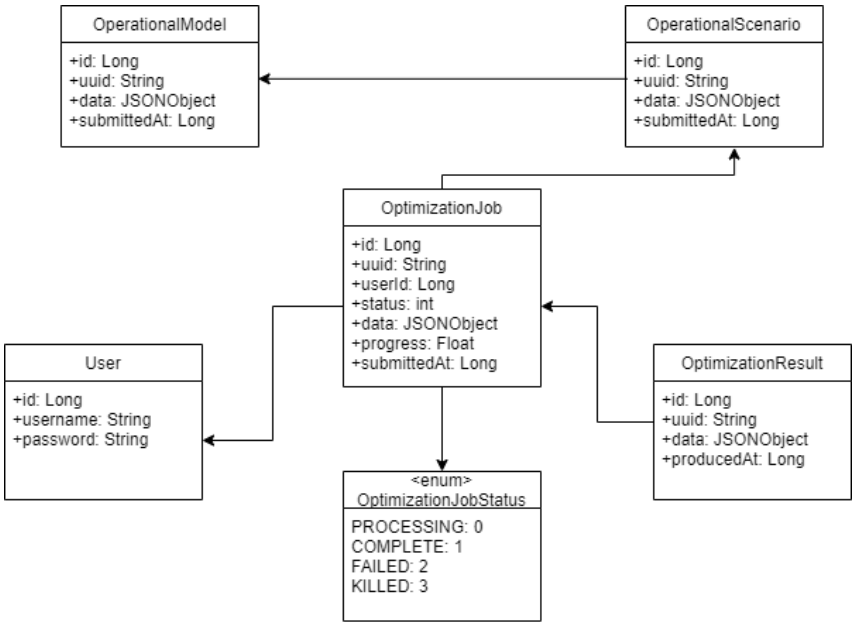

Figure 2. optEngine Data Requirements

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.

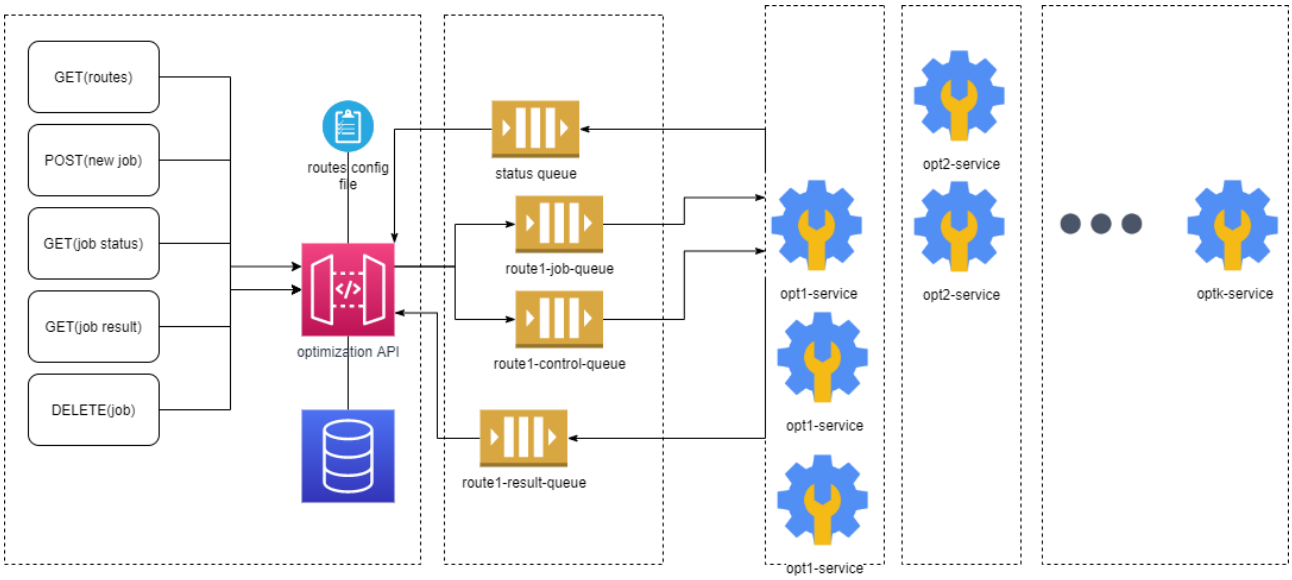

Figure 3. optEngine data Flow

2.2 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

2.3 Web API documentation

2.3.1 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.

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

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

2.3.2. 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	ModelSubmissionDTO	
Description	Contains information about the operational model.	
Attributes	Name	Description
	data	The required data for the operational model. The schema is custom to each type of model.

Example	{
	"data": {
	"empty": true,
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	}
	}	

Name	OperationalScenarioSubmissionDTO	
Description	Contains information about the operational scenario.	
Attributes	Name	Attributes
	Model_uuid	Unique identifier of the call.
	data	The array with the available routes.

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

	<pre> } }</pre>

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	<pre>{
	  "route": "max-flow",
	  "data": {
	  "empty": true,
	  "additionalProp1": {},
	  "additionalProp2": {},
	  "additionalProp3": {}
	  }
	}</pre>	

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	<pre>{
	  "model_uuid":" c11dc261-d8a4-4174-897f-3864defee150",
	  "submitted_at": 1623427969000,
	  "progress": 67.5,
	  "uuid": "c11dc261-d8a4-4174-897f-3864defee150",
	  "status": 0
	}</pre>	

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"
	}	

Name	AdoptApiError	
Description	Contains information about API errors.	
Attributes	Name	Description
	path	The URL.
	message	The error message.
	uuid	The unique identifier of the optimization job.
	status	The http status.

Example	{
	"path": "/opt/job",
	"message": "Internal Server Error",
	"uuid": "c11dc261-d8a4-4174-897f-3864defee150",
	"status": 500
	}	

3. Oil Refineries: Pilot Case by TUPRAS

3.1 Introduction

The production of off-specs LPG may lead to severe losses for refineries, since then the LPG needs to be re-processed, resulting in reduced production efficiency and increased energy costs. Hence, in this section, based on the analysis of the TUPRAS LPG purification process, we present a mathematical programming approach to minimize the energy consumed for planning on-specs LPG production. Specifically, we formulate the problem as a Mixed Integer Linear Program (MILP) that integrates network flow and blending constraints to identify the most energy efficient combination of configurations for all process units of the LPG purification process to achieve on-specs LPG production. To the best of our knowledge, this is the first approach that examines this process as a whole and not just a process unit individually.

The proposed approach can be utilized for recovering from an off-specs situation within a given time horizon, once such a critical situation is identified by FACTLOG anomaly detection, analytics and simulation modules; this is actually the use case that motivates the TUPRAS pilot of FACTLOG. Most importantly, our approach is generic and hence can be applied on any type of industrial process flow that incorporates blending, for which the final product must comply with certain specifications.

Our approach is based upon modelling the different operating conditions (i.e., different levels of temperature, pressure, reflow) of each process unit as corresponding operational scenarios that result in a specific amount of impurity removal, LPG reduction and energy consumed. Note that these operating conditions of each process unit correspond to values that the process engineers can manipulate.

Hence, the Optimization module will be used within FACTLOG to calculate the optimal on-specs recovery plan (i.e., the plan that minimizes energy consumption) by appropriately selecting (and returning) the proper operational scenario for each process unit. This operational scenario will correspond to a specific combination of operating conditions (e.g., temperature, pressure, reflow) for each process unit. Then, the process engineers will manipulate these values (utilizing the corresponding Model Predictive Control (MPC) of the process unit) to apply the changes to the process units of the LPG purification process.

Note that the discretization imposed by utilizing operational scenarios may lead to a huge number of variables, thus affecting solution time. To handle this, a pre-processing step can be utilized that removes scenarios dominated by others, i.e., those that remove less impurities while consuming more energy. In this regard, we have proposed a two-steps procedure in deliverable D5.1. 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 (reduced) operational scenarios are then introduced in the MILP model. This procedure enables us to examine the different combinations of operating conditions in a detailed manner, without affecting much the overall performance of the Optimization module

Note that we have explored this issue computationally in deliverable D5.1, and have shown that by removing dominated operational scenarios, we can substantially reduce the solution space and hence the corresponding solution time for our MILP. Hence, such a computational study is not included in the current deliverable. However, in D5.1 we have also discussed the application of Data Envelopment Analysis assessment (Charnes et al, 1978) for removing dominated operational scenarios. Therefore, we explore this notion in this deliverable.

Further, we extend our approach to account for uncertainty on the level of the input feeds of the process. These values may be unknown, e.g., due to lack of corresponding sensors. We address the uncertainty on the input flow rates (LPG with impurities) by handling them as variables in specific intervals, instead of a constant. On the other hand, we address the uncertainty on the level of impurities in the input feeds by formulating the corresponding constraints as chance constraints. This also enables us to handle the natural variability inherent in the impurity levels of the input feed.

To obtain the operational scenarios for each process unit, we will utilize the work that is being implemented for FACTLOG in creating Machine Learning models that model the behaviour of the process units (in terms of impurity removal and energy consumption) and their integration with the Simulation module. In this manner, we fully utilize the interplay between the different modules of FACTLOG.

Moreover, we have revised our API calls to enable the easy configuration of the Optimization module with respect to the production schema and the corresponding operational scenarios. This is important, since it enables the real-time reconfiguration of the Optimization module with the corresponding operational scenarios that depict the current state of each one of the process units, leading to a state-aware optimization of the LPG purification process.

With respect to deliverable D5.1, this deliverable (D5.2) offers a more detailed presentation of the LPG purification process as well as a more detailed and deeper examination of the corresponding literature, also including newly introduced approaches, such as Chance Constrained Programming. Moreover, it provides a refined, simplified and crisper version of our proposed MILP model and explains how this model can be extended to handle multiple impurities or more complex flow networks. Further, it explores the use of Data Envelopment Analysis in reducing the number of operational scenarios and hence the solution space. Moreover, it explores uncertainty in the level of impurities in the input feed as well as handling the input feed rate. Finally, it proposes the concept of state-aware optimization and explains how this can be implemented by taking advantage of the functionalities that other FACTLOG modules offer.

The remainder of Section 3 is organized as follows. Section 3.2 provides background information on the LPG purification process. Section 3.3 presents related literature. Section 3.4 elaborates on our proposed MILP. More specifically, Section 3.4.1 presents our modelling approach. Section 3.4.2 presents a MILP that incorporates flow and blending constraints to minimize energy consumption while satisfying the corresponding flow, blending and specification constraints. Section 3.4.3 explains how Data Envelopment Analysis can be utilized to reduce the number of operational scenarios. Section 3.5 discusses how we deal with uncertainty on input flow rates. Specifically, the input feed rates of the LPG purification process are considered as variables that receive values in specific intervals (Section 3.5.1), while Chance Constrained Programming (Section 3.5.2) can be

utilized to handle uncertainty with respect to the level of impurities in the input feed. Section 3.6 presents how the concept of state-aware optimization will be implemented within FACTLOG. Finally, Section 3.7 contains the reference list.

3.2 The LPG purification process

Liquified Petroleum Gas (LPG) is a fuel produced as a by-product of various oil refinery processes. The LPG produced by these processes may contain impurities, i.e., Ethane and Naphtha as well as Sulphur compounds. LPG must adhere to certain specifications with respect to the level of these impurities. These may differ for different markets, seasons or applications. Indicative LPG specifications may be 0.5% mol/mol Ethane (i.e., C2), 2% mol/mol Naphtha (i.e., C5 and above) and 50 mg/kg Sulphur as well as specifications for total C2 and C5 content.

Hence, LPG is further processed (i.e., purified) so that the final LPG product meets certain quality specifications with respect to the proportion of impurities within it. In this regard, the purification process of LPG (Bahadori, 2014) involves a network of connected process units of different types. Such units include debutanizers and deethanizers as well as units using organic compounds such as diethanolamine (DEA) to withhold the corresponding substances.

In particular, each input feed flows through a stream of successive process units that are configured accordingly to remove certain impurities. Within this purification process, different streams may be consolidated, in which case the corresponding LPG is blended before entering the next process unit. At the end of the purification process, the LPG of all streams is blended in a single tank. The final LPG product within that tank needs to adhere to the abovementioned quality specifications. Note that, with respect to each process unit of the purification process, different configurations (i.e., settings of temperature, pressure or reflow) correspond to different levels of impurity removal as well as different levels of energy consumption.

A flow network of a typical LPG purification process can be found in Figure 5; this network flow is based upon the corresponding purification process of TUPRAS. The input feed (i.e., LPG containing impurities) comes from various sources, such as Crude Distillation Units (CDU), Hydrocracking (HYC), Fluid Catalytic Cracking (FCC), Delayed Cocker Unit (DCU), Maximum Quality Diesel (MQD) and Platformers. Initially, each input feed is treated to remove carbon-based impurities (i.e., Ethane and Naphtha). Depending on the input feed, this step may require processing from a debutanizer or from both a debutanizer and a deethanizer. Next, the output may also be further processed by an LPG DEA unit to remove Sulphur-based impurities like Hydrogen Sulphur (H_2S) and Mercaptan (CH_4S). At the final stage of the process, all purified LPG flows are blended in the final LPG tank.

With respect to Figure 5, the three CDU input feeds (F1 to F3) are treated by three corresponding debutanizer units (CDU-1 to CDU-3 debutanizers, respectively), whose outputs are subsequently mixed and treated for the removal of Sulphur compounds in a single unit (LPG DEA-1), before flowing in the final LPG tank. The input feeds coming from FCC (F6), DCU (F7) and MQD (F8) processes (F6-f8) are treated in a similar manner in separate streams, though. The HYC input feeds (F4-F5) are treated both by a debutanizer

and deethanizer unit (which may also be placed in reversed order) and then by LPG DEA units. On the other hand, Platformer input feeds (F9-F10) are already processed for Sulphur hence they are only treated by debutanizer units.

Note that the purification process may also be supported by a DEA regeneration unit, which is used to regenerate the diethanolamine amin and collect Sulphur as well as a gas compressor. Note that we have not included such process units in our presentation and the subsequent analysis, as they are not directly involved in the actual purification process. However, it is quite trivial to also account for such unit utilizing our proposed approach.

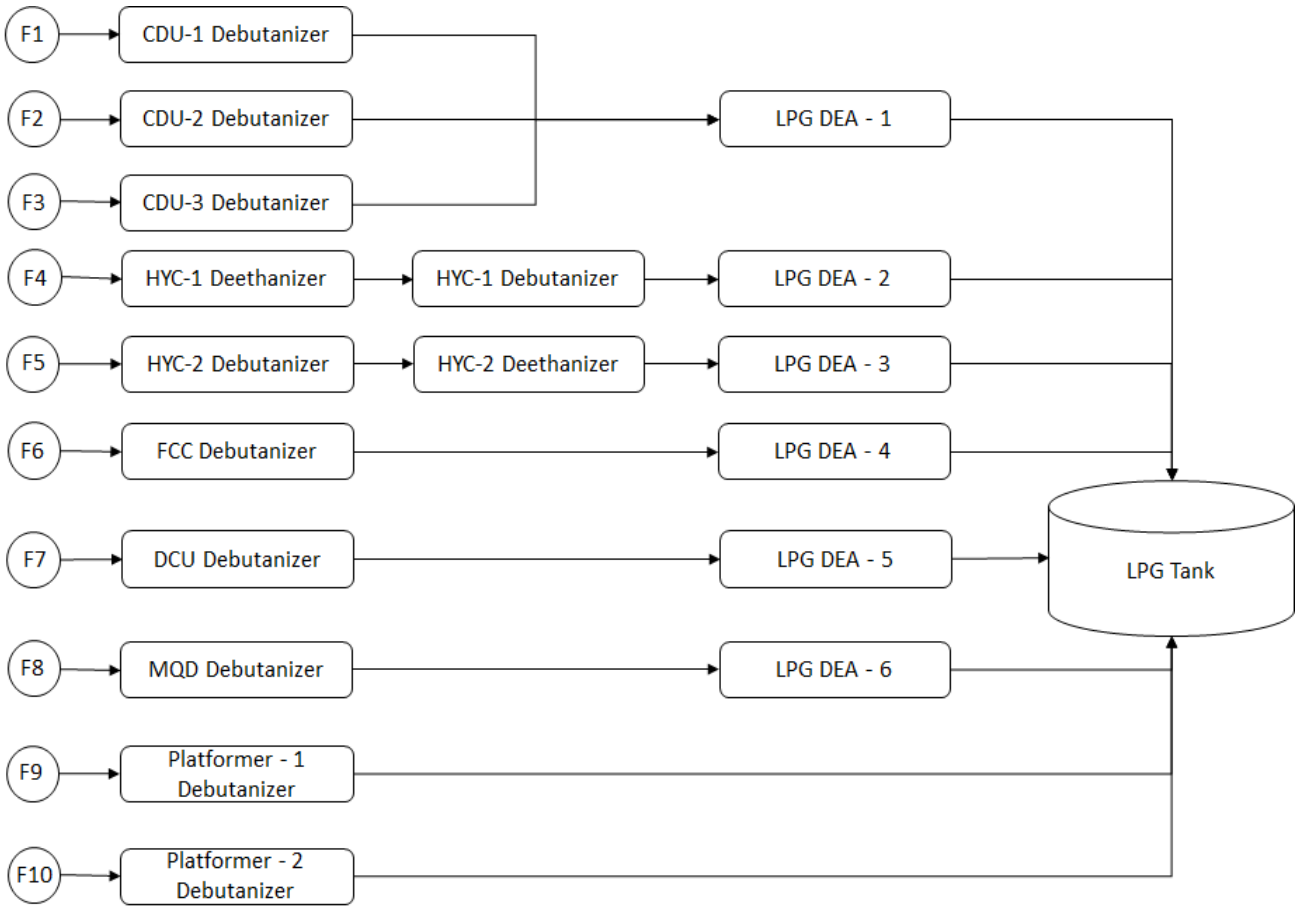

Figure 5: An example of a LPG purification process flow network

3.3 Related literature

Several challenges of the oil refining industry have been met by employing mathematical programming. The optimal topology configuration of petroleum refineries based on MILP is determined in Albahri et al. (2018). 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. Moreover, planning and scheduling issues concerning the petroleum supply network of typical refineries are addressed by utilizing MILP in Kuo and Chang (2008). Pinto and Moro (2000) develop a MILP model to generate a schedule for LPG refinery management to optimize the

selection of storage facilities that are used to receive these products and to feed the product pipeline. Almeida Neto et al. (2000) study a debutanizer unit from the gasoline stream producing LPG, by incorporating linear models to Model Predictive Control (MPC). A MPC for a crude oil preheat and distillation column of a crude oil unit is presented in Kemaloglu et al. (2009). The use of simplified empirical nonlinear process models for CDU and FCC units and accordingly for refinery planning is proposed in Li et al. (2005). In addition, several studies are devoted to the maximization of the production of LPG specifically in the FCC unit. Such studies are based on non-linear optimization models to accommodate the operation of the corresponding MPC, e.g., (de Gouvea and Odloak, 1998; Zanin et al., 2000 and Zanin et al., 2002). Vasconcelos et al. (2005) resort to sequential quadratic programming for the maximization of the LPG and gasoline profit.

A critical review of Natural Gas Liquid (NGL) recovery processes is conducted by Qyyum et al. (2022). The NGL recovery and maximization of plant profitability is formulated as a mixed integer nonlinear programming optimization problem in Murali et al. (2020). The operating conditions of the distillation columns, such as pressure and temperature of feed gas, are considered as decision variables, while the specifications for product impurities are used as constraints in the optimization problem. Nonlinear programming is employed in de Almeida Franco et al. (2020) to minimize the energy consumption for natural gas recovery by meeting safety and quality specifications. The hydrodesulfurization of LPG feedstock by minimizing the energy costs is performed via nonlinear programming in Safari and Vesali-Nase (2019).

Our approach also considers the operating conditions (e.g., temperature, pressure, etc.) of the process units but not each one as a separate decision variable. Instead, it considers them as a combination of operating conditions which may be selected or not, i.e., an operational scenario that removes a specific percentage of impurities and consumes a specific amount of energy. Hence, we address the nonlinearity introduced by temperature, pressure etc. via a binary decision on whether to apply a specific operational scenario or not.

In this regard, our approach follows a different direction. It offers an operational model of the whole LPG purification process, incorporating the way each process unit operates with the structure of the whole flow network and blending constraints for the corresponding quality specifications.

Since the number of processing units is pre-set (according to the LPG production schema of the refinery), the higher the number of the operational scenarios that will be generated the higher the number of variables in our MILP. Thus, in a pre-processing step we reduce the number of operational scenarios by applying Data Envelopment Analysis. In particular, each scenario is viewed as entity under evaluation. Data Envelopment Analysis has been already applied in the context of refineries. The performance of Chinese ethylene production plants is assessed by this method in Han and Geng (2014) and Han et al. (2015). Martin et al. (2016) utilized Data Envelopment Analysis to identify the most sustainable options for a system by considering economic, environmental and social indicators. Their approach is applied to a real case for screening the alternative technologies of electricity generation. Han et al. (2016) proposed an approach for selecting optimal temperature of ethylene cracking furnaces in petrochemical industry. They employed Data Envelopment Analysis to evaluate the operation conditions of ethylene cracking process for different temperature levels. The operation parameters used are the ethylene, propylene, hydrogen, methane, butylene and butane, which considered as outputs, while as input is considered the

feedstock flow. Gonzalez-Garay and Guillén-Gosálbez (2018) incorporated Data Envelopment Analysis in their framework for designing sustainable chemical processes. The method is used for ranking and filtering the process designs obtained as solutions from multi-objective optimization. Their framework is illustrated on the production of methanol from CO_2 and hydrogen. Arabi et al. (2019) employed Data Envelopment Analysis in the context of algae-based biofuel supply chain network to rank the Iranian cities of microalgal harvesting. Dalei and Joshi (2020) used the method for the assessment of Indian oil refineries. Gong et al. (2017a) and (2017b) evaluated different operating conditions in ethylene production process. The operating conditions include the ethylene yield as output while as inputs the fuel gas, high pressure steam, electricity, N_2, compressing gas, recycled water, industrial water and desalted water.

In our approach we account for uncertainty in input feed rates. As the inflow rates (LPG with impurities) are inaccurate, we estimate the bounded intervals in which each of them lies. A common approach to handle interval uncertainties is Interval Linear Programming, see Charnes et al. (1977) and Chinneck and Ramadan (2000). However, we employ a different approach proposed by Despotis and Smirlis (2002) to treat each inflow rate as a variable that lies within a bounded interval. In this way, we also provide each unit with the flexibility to determine the optimal level of the inflow rate for the operational scenarios given.

The uncertainty in the input feed impurities is handled by employing Chance Constrained Programming (CCP). CCP has been widely utilized to solve optimization problems under uncertainty. The constraints with stochastic parameters are formulated probabilistically, in the sense that they should be satisfied with a given probability level. CCP was originally introduced in Charnes and Cooper (1959) for single chance constraints, i.e., chance constraints that should be probabilistically satisfied individually. A variant that accommodates joint constraints is proposed by Miller and Wagner (1965). These chance constraints should be probabilistically satisfied jointly, given a single probability level.

Contrary to conventional optimization problems there is not any general method for solving CCP problems. The strategy is to transform the chance constraints to deterministic ones. This entails the calculation of the prescribed probability. However, the probability distribution may be unknown, or difficulties may arise on calculating probabilities because of multi-dimensional integration. Thus, the solution method for CCP problems depends on i) the form of the probabilistic constraints (joint or individual), ii) the distributions of the random parameters and iii) the nature of the constraints (linear, convex, etc.). Theoretical and computational aspects of CCP can be found in (Prekopa, 1995 and Ruszczynski and Shapiro, 2003).

A comprehensive summary of theoretical developments and practical applications of CCP in the process industries is provided in Li et al. (2008). CCP has been already applied to refinery planning, for instance to estimate the expectation of plant revenues in Li et al. (2004). A literature review of refinery planning models that deal with uncertainty can be found in Leiras et al. (2011). Probabilistic constraints are used in Henrion and Möller (2003) to extract feed from the tank of a continuous distillation process in which the rate of inflows is unknown. Also, CCP was applied to the problem of a distillation process that separates methanol-water mixture under uncertain inflows in Li et al. (2002). A CCP approach to handle the uncertainty in feed flow rate of a gas processing plant and maximize the overall profit is proposed in Mesfin and Shuhaimi (2010). The proposed approach was extended in (Getu et al., 2012; 2013) to incorporate the uncertainty effect from the plant outlet side. This

approach was also applied in Getu et al. (2015) to examine six different process schemes for NGL recovery using feed conditions such as temperature, pressure, flowrate, etc. A MILP is developed in (Al-Qahtani and Elkamel, 2008) to examine the process network integration alternatives in a multisite petroleum refinery system with the aim to minimize the total cost. This model is extended in (Al-Qahtani and Elkamel, 2010) to consider uncertainty in raw materials and final product prices as well as products demand. The parameters with uncertainty are raw materials, product prices, and market demand. (Salas et al., 2021) proposed a multi-objective framework for determining the optimal operating conditions of an NGL recovery unit. The two objectives are the annual profitability of the unit and the concentration of methane in the NGL product stream. The market uncertainty of the prices of the products and the costs of raw materials is also incorporated in the model.

We utilize CCP to extend our proposed MILP to handle the uncertainty that permeates the inflows in terms of the impurities included within LPG entering the purification process. In this regard, we model these parameters as random variables and employ CCP to incorporate them into our MILP to obtain solutions within a specific probability level.

3.4 MILP for on-specs LPG production and recovery

3.4.1 Modelling approach

A key issue for our proposed approach lies in the way we handle the operating conditions of each process unit. Each process unit is modelled as a process that transforms inputs into outputs. In this regard, the operating conditions of each process unit (e.g., temperature, pressure, etc.) constitute the inputs, while the resulting reduced LPG flow, removed impurities and energy consumed constitute the outputs. The mapping of all possible inputs to their corresponding outputs for a specific process unit constitute its operational scenarios. That is, each operational scenario is fully specified by (i) the operating conditions of the process unit (pressure, temperature, etc.), (ii) the reduction in the flow rate of LPG and impurities of outputs from the unit (products) and (iii) the corresponding energy consumption.

Hence, we have a specific operational scenario for each combination of operating conditions of a given process unit. For instance, a debutanizer receives a specific feed and applies temperature 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 operating conditions that may be applied (e.g., higher/lower temperature with different levels of pressure) result in different outcomes (level of LPG reduction and impurities removed) and accordingly to different energy consumption levels.

Higher energy consumption leads to larger impurity removal, hence on-specs production requires more energy. However, the optimal quantity of energy required and the corresponding operating conditions are unknown. Therefore, we build our MILP model with the aim to minimize energy consumption while satisfying the production specifications of LPG and by incorporating the operational scenarios of each process unit.

We model the LPG purification process via a MILP 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.

These operational scenarios can be created using either a model- or a data-driven approach. In the former, a physical model of the process transformation is used to simulate the different levels of impurity removal and energy consumption for each combination of operating conditions. In the latter, a Machine Learning model can be utilized, provided that the corresponding sensor data is available Rožanec et al. (2021).

Note that this modelling approach has enabled us to avoid directly incorporating temperature, pressure, etc. and their relation to impurity removal and energy consumption as variables in the model, and hence to avoid introducing the nonlinear relationships that these impose. That is, we have removed the nonlinearity from the model in a rather intuitive manner, by introducing binary decision variables (based on the operational scenarios) that model the actual decisions that a control/process engineer needs to take when planning for on-specs LPG production, i.e., how to change the operating conditions of each process unit.

3.4.2 The proposed MILP

In what follows, we provide the notation of the sets, constants and variables that are employed in our proposed MILP. The quantities are measured in m^3, the time intervals in hours flow rates in m^3/hour and energy consumption rate in kJ/hour. The input feeds are assumed stable for the whole period. For simplicity of presentation, we present our model with only one impurity (i.e., C2). The extension of this model to multiple impurities is discussed in the end of this subsection.

Let us now formally define our model. To enable easier understanding, sets and constants are provided with capital letters, variables with small ones.

Consider the flow network $G(V, E)$, portrayed in **Errore. L'origine riferimento non è stata trovata.**, in which each input feed (V_{in}), process unit (V_{pr}) and output tank (V_{out}) is a node $i \in V = V_{in} \cup V_{proc} \cup V_{out}$, and any connection between two such nodes is an edge $(i, j) \in E$. For simplicity of exposition, we assume that any path in G starts from a node in V_{in} and ends with a node in V_{out}. Let CAP_{ij} denote the flow rate capacity of $(i, j) \in E$, N_i^- the set of nodes to which i sends flow (i.e., $j \in N_i^-: (i, j) \in E, i \in V_{in} \cup V_{proc}$), and $N_i^+$the set of nodes from which i receives flow (i.e., $k \in N_i^+: (k, i) \in E, i \in V_{out} \cup V_{proc}$).

With respect to input feeds, each node $i \in V_{in}$ has a given input flow rate IF_i which also contains a flow rate $IC2_i$ of impurity C2 (this may be provided as a percentage of IF_i but can be easily converted to flow rate). With respect to the output tank(s), for each node $i \in V_{out}$, let Q_i^{total} be the total capacity of that tank, Q_i^{start} the quantity of the tank at the start of the purification process and $QC2_i^{start}$ the quantity of C2 within that tank (again, this may be provided as a percentage of Q_i^{start} but can be easily converted to quantity).

With respect to process units, each node $i \in V_{proc}$ is associated with a set of operational scenarios S_i. Each $i \in V_{proc}$, using $s \in S_i$, is associated with the corresponding energy consumption rate E_i^s. Also, let PF_{ij}^s be the percentage of total flow rate that $i \in V_{proc}$ sends to $j \in N_i^-$ under $s \in S_i$. Hence, $1 - PF_{ij}^s$ corresponds to the percentage flow rate removed by

the purification process from the stream leading to the final LPG tank. In a similar fashion, let $PC2_{ij}^{s}$ be the percentage of C2 flow rate that $i \in V_{proc}$ sends to $j \in N_i^-$ under $s \in S_i$. In this regard, $PC2_{ij}^{s} = 100\%$ would mean that no C2 is removed from the LPG, while $PC2_{ij}^{s} = 0\%$ would mean that all C2 is removed from the LPG.

Further, let H be the time horizon for which LPG production needs to be scheduled and SP_{C2} the percentage specification for C2, i.e., the maximum allowed percentage of C2 in the final tank, in order for the LPG to be on-specs.

Let us now define the corresponding variables. With respect to process unit $i \in V_{proc}$ running operational scenario $s \in S_i$, let $x_i^s \in \{0,1\}$ denote the decision variable that shows whether i runs s ($x_i^s = 1$) or not ($x_i^s = 0$). Accordingly, $f_{ij}^{s} \in \mathrm{R}$ denotes the total flow rate of $i \in V_{proc}$ running $s \in S_i$ towards node j $\in N_i^-$. Moreover, $f_{ij}^{*} \in \mathrm{R}$ denotes the total flow rate of $i \in V_{in} \cup V_{proc}$ towards node j $\in N_i^-$. Hence, for input feed $i \in V_{in}$, f_{ij}^{*} equals the total flow rated of that input feed, while for process unit $i \in V_{proc}$, f_{ij}^{*} equals the total flow rate of the chosen operational scenario for that process unit. In a similar fashion, $fC2_{ij}^{s} \in \mathrm{R}$ denotes the C2 flow rate of $i \in V_{proc}$ running $s \in S_i$ towards node j $\in N_i^-$ and $f_{ij}^{*} \in \mathrm{R}$ the total flow rate of $i \in V_{in} \cup V_{proc}$ towards node j $\in N_i^-$. Finally, let $q_i \in \mathrm{R}$ be the quantity of LPG in the final tank $i \in V_{out}$ at the end of time horizon H and $qC2_i \in \mathrm{R}$ be the quantity of C2 in the final tank $i \in V_{out}$ at the end of time horizon H.

Note that for convenience, all notation is also provided in Table 1 (sets), Table 2 (constants) and Table 3 (variables).

Table 1. List of sets

V_{in}	Set of nodes corresponding to input feeds
V_{proc}	Set of nodes corresponding to process units
V_{out}	Set of nodes corresponding to output tanks
N_i^-	Set of nodes to which i sends flow, $i \in V_{in} \cup V_{proc}$
N_i^+	Set of nodes from which i receives flow, $i \in V_{out} \cup V_{proc}$
S_i	Set of operational scenarios for node $i, \forall i \in V_{proc}$

Table 2. List of constants

H	Time horizon for recovery
SP_{C2}	Specifications (%) for C2 in final LPG tanks
IF_i	Total flow rate of $i \in V_{in}$ (LPG with impurities)
$IC2_i$	Flow rate of C2 within LPG of $i \in V_{in}$
CAP_{ij}	Capacity of flow rate $i \in V_{pr}$ can send to $j \in N_i^-$
Q_i^{total}	Total quantity capacity of LPG in $i \in V_{out}$
Q_i^{start}	Current quantity of LPG in $i \in V_{out}$
$QC2_i^{start}$	Current perc. of flow rate of C2 that $i \in V_{pr}$ sends to $j \in N_i^-$ under $s \in S_i$ for a time unit
E_i^s	Energy consumption rate of node $i \in V_{pr}$ using $s \in S_i$ for a time unit

PF_{ij}^{s}	Perc. of total flow rate that $i \in V_{pr}$ sends to $j \in N_i^-$ under $s \in S_i$ for a time unit (the rest corresponds to removed impurities)
$PC2_{ij}^{s}$	Perc. of flow rate of C2 that $i \in V_{pr}$ sends to $j \in N_i^-$ under $s \in S_i$ for a time unit ($1 - PC2_{ij}^{s}$ corresponds to removed impurities)

Table 3. List of variables

$x_i^s \in \{0,1\}$	Node $i \in V_{proc}$ runs $s \in S_i$ ($x_i^s = 1$) or not ($x_i^s = 0$)
$f_{ij}^s \in R_0^+$	Total flow rate of $i \in V_{proc}$ running operational scenario $s \in S_i$ towards j $\in N_i^-$.
$f_{ij}^* \in R_0^+$	Total flow rate of $i \in V_{in} \cup V_{proc}$ towards $j \in N_i^-$
$fC2_{ij}^s \in R_0^+$	C2 flow rate of $i \in V_{proc}$ running operational scenario $s \in S_i$ towards $j \in N_i^-$.
$fC2_{ij}^* \in R_0^+$	C2 flow rate of $i \in V_{in} \cup V_{proc}$ towards $j \in N_i^-$
$q_i \in R_0^+$	Quantity of LPG in $i \in V_{out}$ at the end of time horizon H
$qC2_i \in R_0^+$	Quantity of C2 in $i \in V_{out}$ at the end of time horizon H

The proposed MILP is provided below.

$$\min \sum_{i \in V_{proc}} \sum_{s \in S_i} H \cdot E_i^s \cdot x_i^s$$

$$\sum_{s \in S_i} x_i^s = 1, \qquad \forall i \in V_{proc} \qquad (3.1)$$

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

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

$$f_{ij}^s \le CAP_i \cdot x_i^s, \qquad \forall i \in V_{proc}, s \in S_i, j \in N_i^- \qquad (3.4)$$

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

$$q_i = Q_i^{start} + H \cdot \sum_{k \in N_i^+} f_{ki}^*, \qquad \forall i \in V_{out} \qquad (3.6)$$

$$q_i \le Q_i^{total}, \qquad \forall i \in V_{out} \qquad (3.7)$$

$$fC2_{ij}^* \ge IC2_i, \qquad \forall i \in V_{in}, j \in N_i^- \qquad (3.8)$$

$$fC2_{ij}^* = \sum_{s \in S_i} fC2_{ij}^s, \qquad \forall i \in V_{proc}, j \in N_i^- \qquad (3.9)$$

$$fC2_{ij}^s \le f_{ij}^s, \qquad \forall i \in V_{proc}, s \in S_i, j \in N_i^- \qquad (3.10)$$

$$\sum_{k \in N_i^+} fC2_{ki}^* = \sum_{s \in S_i} \frac{1}{PC2_{ij}^s} \cdot f_{ij}^s, \qquad \forall i \in V_{proc}, j \in N_i^- \qquad (3.11)$$

$$qC2_i = QC2_i^{start} + H \cdot \sum_{k \in N_i^+} fC2_{ki}^*, \qquad \forall i \in V_{out} \qquad (3.12)$$

$$qC2_i \leq SP_{C2} \cdot q_i, \quad \forall i \in V_{out} \tag{3.13}$$

$$x_i^s \in \{0,1\}, \quad \forall i \in V_{proc}, s \in S_i \tag{3.14}$$

$$f_{ij}^s, fC2_{ij}^s \geq 0, \quad \forall i \in V_{proc}, s \in S_i, j \in N_i^- \tag{3.15}$$

$$f_{ij}^*, fC2_{ij}^* \geq 0, \quad \forall i \in V_{in} \cup V_{proc}, j \in N_i^- \tag{3.16}$$

$$q_i, qC2_i \geq 0, \quad \forall i \in V_{out} \tag{3.17}$$

The objective function minimizes total energy consumption given the operational scenario selected for each process unit. By constraint (3.1), exactly one operational scenario is selected for each process unit. Constraint (3.2) introduces the input feed to the subsequent process unit. Constraint (3.3) models the flow rate of LPG from each process unit to the next node (i.e., process unit or output tank). Note that this flow rate will be equal to flow rate of the corresponding selected scenario, since the flow rate variable f_{ij}^s for any operational scenario not selected (i.e., for any x_i^s=0) is pushed to 0 by constraint (3.4). The flow rate that a process unit receives from its predecessors is calculated by constraints (3.5), reduced by the reduction of flow at each predecessor node. At the end of the time horizon, the quantity of LPG that is contained in the final LPG tank is calculated by constraint (3.6); this quantity cannot exceed the capacity of the corresponding output tank(s), by constraint (3.7).

While constraint (1) handles the operational scenario selection and constraints (3.2)-(3.7) the flow of LPG, constraints (3.8)-(3.13) handle C2 impurity removal of the purification process. In this regard, constraints (3.8)-(3.12) are the C2 impurity flow counterparts of constraints (3.2)-(3.6), while constraint (3.13) imposes the specification objective for C2 in the final LPG tank(s). Note that the use of '≥' in constraint (3.8), instead of an equality, does not affect the final result, since the minimization objective function paired with constraint (3.13) pushes constraint (3.8) to be satisfied as an equality. Constraints (3.14) and (3.15)-(3.17) are binary and non-negativity constraints for the corresponding variables, respectively.

In this section, we have presented a MILP for on-specs LPG production. The proposed model (3.1)-(3.15) can be extended to handle multiple impurities. For example, to handle C5 constraints, we can include C5 variables ($fC5_{ij}^s, fC5_{ij}^*, qC5_i$) and add a set of corresponding constraints similar to the ones for C2 (i.e., constraints (3.8)-(3.13)). Similarly, we could also include Sulphur or any other impurity required. Moreover, to handle specifications on the total amount of different types of impurities (e.g., total amount of C2 and C5 included, say SP_{C2+C5}), we can extend constraint (3.13) as follows:

$$qC2_i + qC5_i \leq SP_{C2+C5} \cdot q_i, \forall i \in V_{out} \tag{3.18}$$

Further, the process model that the Simulation tool is based upon and uses (and the Optimization model consumes and utilizes) also includes a specific type of unit to unify flows, namely junctions. For example, in **Errore. L'origine riferimento non è stata trovata.**, the flows that leave the three CDU debutanizers are blended before entering the corresponding LPG DEA unit. This blending is done via a junction. To incorporate junctions within our model, we consider them as process units, i.e., all junctions are considered as nodes included in V_{proc}. That is, we consider junctions to be process units. Then, it suffices to define only one operational scenario for each such node (i.e., junction), one that does not

reduce LPG or impurities at all and does not consume any energy. Hence, for each junction $i_{jun} \in V_{proc}$, we consider only one operational scenario, i.e., $s \in S_{i_{jun}}$ such that $\left|S_{i_{jun}}\right| = 1$, for which $x^s_{i_{jun}} = 1, PF^s_{i_{jun}j} = 1$ and $PC2^s_{i_{jun}j} = 1$ (and similarly for any other impurity, signifying that no LPG or impurity is removed) and $E^s_{i_{jun}} = 0$, since junctions do not consume any energy.

Note that we have presented our approach in terms of on-specs LPG production. However, our approach can be easily utilized once an off-specs situation has been identified. Our model can then be used to provide a plan for on-specs recovery within a given time horizon and with the minimum energy consumption. To do so, we only need to initialize our output tank constants (i.e., the amount of LPG and corresponding impurities in the output LPG tank) at the level they currently are, i.e., incorporating the off-specs impurity levels.

More importantly, our model is presented in a generic manner and thus can handle any amount of input feeds or output tanks, and any flow network of process units that performs a similar operation, removing impurities and blending different streams. Similarly, it can handle different sources of uncertainty stemming from the input feed.

3.4.3 Removing operational scenarios via Data Envelopment Analysis

In deliverable D5.1, we have proposed a two-step pre-processing procedure for reducing the number of operational scenarios and hence reducing the size of the MILP problem (i.e., the number of variables) and the solution space. This leads to important computational savings and improvement of the time performance of the solution method. In our two-step procedure, we first calculate the convex hull of the operational scenarios by employing the quickhull algorithm (Barber et al 1996) to identify the extreme operational scenarios in the convex hull that includes all of them. Next, we compare them based on dominance relationships to derive only the dominant ones. These (reduced) operational scenarios are then introduced in the MILP model. In this deliverable, we explore the use of Data Envelopment Analysis (Despotis, 2005) for reducing the number of operational scenarios.

For this purpose, we consider operational scenarios as entities to be evaluated. Notice that each operational scenario *i* includes five measures-criteria (fi1-fi5), i.e., the energy consumption, the reduction rate of C2, the reduction rate of C5, the reduction rate of Sulphur and the reduction rate of LPG. We use these values to create an aggregated measure for each operational scenario, i.e., a score. In this regard, we employ a linear programming approach, which is based on Data Envelopment Analysis, to derive the weights for the aggregation. In particular, for each specific scenario *i* we calculate its score as the weighted sum $h_i = uf_i$, where $f_i = (f_{i1}, f_{i2}, \dots, Y_{i5})^T$ denotes the vector of the m values of the and $u = (u_1, u_2, \dots, u_5)$ denotes the vector of the variables used as weights. A common weighting scheme for the aggregation is obtained from the following model:

$$min \sum_{i=1}^{|V_{in}|} d_i \qquad (3.19)$$

$$s.t.$$

$$uf_i + d_i = 1, \;\; i = 1, \dots, n$$

$$u \geq 0, d_i \geq 0$$

We assume an upper bound for the score of each scenario *i* which is set to 1, i.e., $uf_i \leq 1$. In model (3.19) the deviations d_i of the performances of all scenarios from the upper bound are simultaneously minimized. As lower levels of energy consumption, LPG reduction, etc. are desired, these criteria are modified accordingly to be appropriate for maximization. The optimal solution of the linear model (3.19) provides an aggregation scheme as well as it allows for ranking the scenarios. Our procedure may serve as a filter for the MILP model by excluding the operational scenarios with $h_i < 1$. Hence, we can use this method by simply creating and solving (via any Linear Programming solver) the corresponding linear model (3.19) for the operational scenarios.

3.5. Handling input uncertainty

3.5.1 Handling uncertainty in the input feed rate

As noticed, there is uncertainty in the total input flow rate (LPG with impurities), feeds F1-F10 in Figure 5. Hence, we do not consider the inflows rate as constants anymore. The lower and upper bound that each inflow rate can vary are estimated or given by the analyst. Now, the analyst instead of indicating a specific input flow rate for each unit, can safer propose an interval that its value can vary. Following Despotis and Smirlis (2002), we address this kind of uncertainty by expressing each inflow rate IF_i as a variable within the bounded interval $[IF_i^L, IF_i^U]$:

$$IF_i = IF_i^L + \gamma\left(IF_i^U - IF_i^L\right), \qquad \forall i \in V_{in}, j \in N_i^- \qquad (3.20)$$

The constraints (3.2) that incorporate the input flow rates are converted to the following ones:

$$f_{ij}^* = IF_i^L + \gamma\left(IF_i^U - IF_i^L\right), \qquad \forall i \in V_{in}, j \in N_i^- \qquad (3.21)$$

The new variable γ represents the additional input flow rate IF_i within the interval $[IF_i^L, IF_i^U]$. The previous form of the proposed MILP is obtained when the lower and upper bounds coincide ($IF_i^L = IF_i^U$).

The adopted approach enables us to examine fewer operating schemes concerning the inflow rates. It is not required anymore to explicitly evaluate every different possible value of the inflow rates as optimization provides their optimal values within a given range. In addition, testing specific values for inflow rates may be restrictive and render the proposed MILP infeasible. The model is more flexible to avoid such a situation with inflow rates being

variables. Finally, we grant the flexibility to each process unit i ∈ V_{in} to decide the optimal level of the inflow rate that accepts for the operational scenarios under evaluation.

3.5.2 Handling uncertainty in the input feed impurities

The amount of C2 within LPG in the input feed may be uncertain due to the natural variability of the previous processes and the lack of sensors at the beginning of the purification process. Thus, we may wish to treat each $IC2_i$ as an uncertain parameter, i.e., a continuous random variable. These uncertain parameters appear on the right-hand side of the linear constraints (3.8) that correspond to the C2 input feeds. Thus, to take into account the uncertainty, constraints (3.8) may be expressed in the following probabilistic form:

$$P\{fC2_{ij}^{*} \geq IC2_i\} \geq \alpha_i \quad ,\forall\, i \in V_{in}, j \in N_i^{-} \tag{3.22}$$

Many single chance constraints can be individually incorporated into a problem (Birge, 1997). In this regard, each input flow $IC2_i$ may stem from several independent sources. In such case, the input flows are independent, and we can formulate an individual chance constraint for each $IC2_i$. Each chance constraint (3.22) requires the corresponding original constraint in (3.8) to be satisfied within a prescribed probability. The probability that the original constraint i will hold must be at least αi, with the desired confidence level α_i ∈ (0,1] defined by the user. Usually, α_i is selected quite close to 1, as the higher α_i, the more reliable the modeled LPG purification process. However, higher values of α_i lead to shrinkage of the feasible solution space (Henrion et al., 2001).

The underlying one-dimensional distribution of each random parameter $IC2_i$ is required for solving the MILP that incorporates the chance constraints. The individual chance constraints (3.22) can be converted to equivalent deterministic ones by employing the cumulative distribution function Φ_i^{-1} of each random parameter $IC2_i$:

$$fC2_{ij}^{*} \geq \Phi_i^{-1}(\alpha_i) \quad ,\forall\, i \in V_{in}, j \in N_i^{-} \tag{3.23}$$

The α-quantile of each distribution Φ_i is represented by $\Phi_i^{-1}(\alpha_i)$ in constraints (3.23), see (Birge and Louveaux, 2011).

The distribution of each input flow rate $IC2_i$ can be identified by historical data and through sampling from the LPG process. However, if the input flow entering the purification process of LPG is derived from the aggregation of many other independent flows stemming from different sources, it can be assumed that each independent input flow rate $IC2_i$ follows a Normal distribution $N(\mu_i,\sigma_i)$, since the rate of the overall inflows in a continuous distillation process may be considered as a Gaussian process (Henrion and Möller 2003). Then, for each inflow rate $IC2_i$, the parameters of distribution $N(\mu_i,\sigma_i)$ can be estimated from historical observations.

The probability for each constraint (3.23) to be satisfied is at least α_i while it will be exactly α_i if at optimality is satisfied with equality. Overall, the on-specs plan for LPG production will be achievable if all constraints (3.23) are satisfied. Hence, as the input flow rates are independent, the joint probability that all constraints (3.23) are satisfied is at least $\prod_{i=1}^{|V_{in}|} \alpha_i$.

3.6 State-aware optimization

In this section, we explain how the FACTLOG approach can be utilized to enable state-aware optimization of the on-specs LPG production or recovery process. In this regard, we first explain how operational scenarios are produced for each process unit and then present how our corresponding FACTLOG implementation enables this.

To obtain the operational scenarios for each process unit, we will utilize the work that is being implemented for FACTLOG in terms of the Analytics module, the Simulation module and the FACTLOG platform. More specifically, based on historical data from each process unit, Machine Learning models are created (by the Analytics module) that model the behaviour of each process unit (in terms of impurity removal, LPG reduction and energy consumption). These models are then integrated within the Simulation module, which can utilize them either to evaluate the performance of the LPG purification process as a whole or of each process unit separately. For the latter, the Simulation module has included a functionality to produce all possible operational scenarios for any process unit of the TUPRAS LPG purification process with a given step for each corresponding operating condition. These operational scenarios are then transferred via the FACTLOG platform to the Optimization module to be consumed. The optimization module utilizes them to model the corresponding on-specs recovery problem (i.e., the operational scenarios for each process unit). In this manner, we fully utilize the interplay between the different modules of FACTLOG (Figure 6).

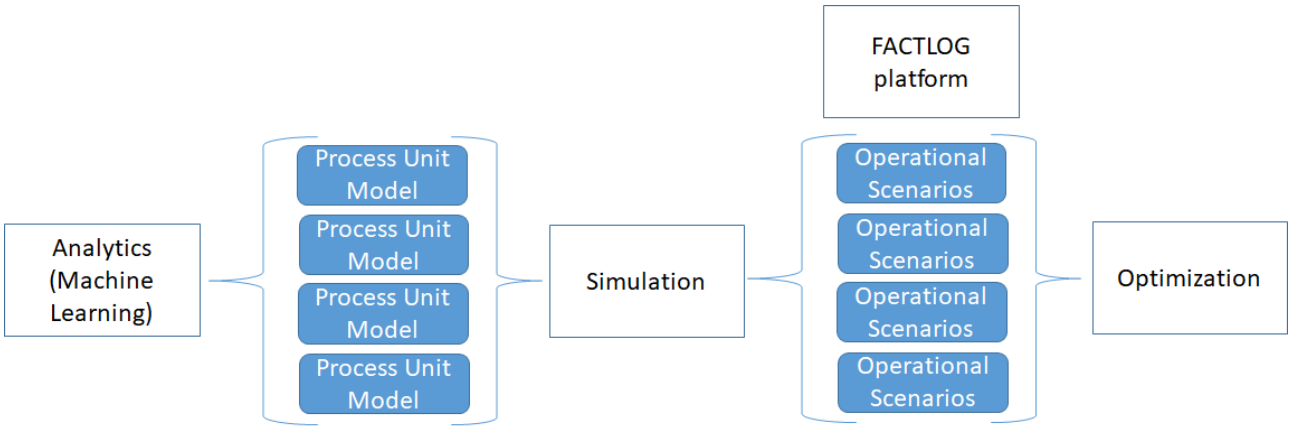

Figure 6. Collaboration of Analytics, Simulation and Optimization modules for operational scenarios

Note that the pre-processing step of reducing the number of operational scenarios can be integrated either within the Optimization module or within the Simulation module. In the former case, the pre-processing step runs as soon as these operational scenarios are received from the FACTLOG platform and before formulating the problem of on-specs recovery. In the latter case, it runs within the Simulation module, after producing the operational scenarios and before sending them to the FACTLOG platform. It must be noted that the latter reduces the amount of data being transferred between modules. Nevertheless, in both cases, the resulting set of operational scenarios would be the same.

By implementing the above collaboration of the Analytics, the Simulation and the Optimization module (via the FACTLOG platform) in a real-time fashion, we can achieve state-aware optimization, i.e., optimization that is dynamic and incorporates the actual current situation of all production process units.

More specifically, once an off-specs situation is identified by FACTLOG (i.e., utilizing its anomaly detection and simulation functionalities), the state-aware optimization process can be initialized by FACTLOG. This would include feeding the Machine Learning models (produced by the Analytics module and consumed by Simulation) with the current real-time dataset of all the process units. The Simulation module then uses this data as input for its underlying models to simulate the current behaviour of each process unit (i.e., the way it currently operates in terms of removing impurities, consuming energy, etc.). Computationally, this does not create any issues, since these models are already trained and Simulation only performs scoring by giving them the corresponding dataset.

The produced operational scenarios then depict the current state of each process unit and of the LPG purification process as a whole, since being created based on the current data of the process. Hence, by consuming these operational scenarios, the Optimization module obtains a model of production which is state-aware, i.e., it is produced taking into account the current state of the process units and the way they operate. Hence, the resulting on-specs recovery plan incorporates the current state of the process units. This is really important, since, for an off-specs situation to occur, it must be the case that one or more process units do not operate as they were supposed to.

4. Textile Industry: Pilot Case by PIACENZA

4.1 Introduction

Background. Manufacturers continuously aim at improving their operations regarding supply, transportation and production, with the aid of cutting-edge technology tools. Within the Industry 4.0 discourse, increased data availability urges stakeholders to expect more efficient solutions to crucial fields of the production chain, including the long-standing aspect of efficient production schedules. That is, scheduling algorithms Brucker (1999) remain a viable and effective tool to improve productivity under challenging, tight and evolving restrictions. These usually refer to the time needed to prepare the machines so as to process a given order, the speed of each machine, the capability of splitting orders into parts, precedence relations among orders and limited availability of resources that facilitate the production process.

Our work is focused on the weaving scheduling of PIACENZA, a textile enterprise in north Italy that manufactures woolen 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. Under these restrictions, different optimization criteria may be used to obtain the final schedule, driven by the completion time(s) of the schedule (jobs) or to the delay beyond due date some orders may experience. Therefore, depending on the existence and tightness of deadlines, we may opt for minimizing either the makespan (no or very loose deadlines) or the total (weighted) tardiness (strict or tight deadlines).

Literature review. We, distinguish previous literature works based on the objective that they try to minimize. For makespan minimization, the work of Kim et al. (2004) provides a two-stage algorithm. The identical-machine scheduling problem with job-splitting and sequence-dependent setup times is addressed by Yalaoui et al. (2003) but only through a heuristic algorithm. For unrelated machines, makespan minimization with sequence-dependent setup times and job splitting is tackled in Eroglou et al. (2014) by employing a genetic algorithm and in Rosales et. al (2015) and by exact formulation but a meta-heuristic approach. Stronger LP Relaxations on unrelated machines with job-splitting are shown in Correa et al. (2015), while Correa et al. (2016) examines approximation algorithms combining job splitting and machine-dependent setups on all parallel machine environments but aiming to minimize the weighted sum of completion times. As the MILP formulations grow large even for medium-sized instances, exact methods relying on a decomposition of the problem or alternative such models have drawn attention recently. The work of Tran et al. (2016) attempts to address unrelated machines with machine and sequence-dependent setup times by using logic-based Benders decomposition and branch-and-check. The same problem variation is examined by Peyro et al. (2019), who provide a relatively efficient MILP formulation and a heuristic algorithm.

In terms of tardiness related objectives, Kim et al. (2020) presents a MILP for the total weighted tardiness minimization of a job shop scheduling problem on parallel identical machines. Setup times are not sequence-dependent, but a maximum allowed tardiness

increases the complexity of the problem. The formulation of Maecker & Shen (2020) exploits the properties of identical machines to consider variables without machines-related indices. Even though the main core of the paper is based on metaheuristic methods, the proposed MILP is also tested on small instances. Ozturk & Chu (2018) study the properties of parallel identical machines environments to deduct dominance rules for batch creation of due-related scheduling problems. The lower bounds, which are implied by these rules, prune branches of the tree to speed up the performance of Branch-and-Bound algorithms. On the other hand, MILP formulations seem to be inapplicable on unrelated machines environments. The study of Bektur & Sarac (2019) is an attempt to extend the popular makespan minimization problem to the weighted tardiness minimization problem. The constructed MILP formulation is tested on a minimal dataset of 10 jobs and 2 unrelated machines. A bi-objective problem (C_{max} and $\sum_j Tj$) is solved by a family of exact, heuristic and metaheuristic methods, as presented by Moser et al. (2021). The generated datasets concern a range of 20-1000 jobs and 1-30 machines. Even though the size of the large datasets is remarkable, the exact methods are applied on the 25 smallest instances. Last, Cheng & Huang (2017) study the minimization of earliness/tardiness problem on unrelated machines, however their proposed algorithm outperforms the respective MILP.

Our contribution. We advance current literature in three main areas. First, as mentioned above, our work has an important impact in a typical textile production. Weaving scheduling problem has been well-studied, achieving exact polynomial time algorithms for special cases where setup times are independent and job splitting is relaxed to preemption, under the goal to minimise the maximum weighted tardiness (Serafini, 1996). Efficient heuristics have been proposed for more general cases where the authors apply genetic algorithms to tackle large real-life instances on unrelated looms under sequence-dependent setup times (Eroglou et al., 2014), job splitting and machine eligibility constraints (Eroglou et al., 2017), with the goal to minimise the makespan of the schedule.

Secondly, in terms of the makespan minimization, we solve the PMS problem on unrelated machines with sequence-dependent setup times, job splitting and resource constraints. Simpler variants of our problem have recently been studied, e.g., Lee et. al (2020) consider identical machines, job splitting, multiple setup resources, fixed (independent) setup times for each job, where the objective is the standard makespan minimization; in the same work, the case of sequence-dependent setup times and unrelated machines is referred as an open problem. This simplified version of our problem when solved by thereby suggested MILP formulation is limited to instances with up to 6 jobs, thus a heuristic is developed to solve instances of up to 100 jobs and 20 machines. The work of Kim et al. (2021) examines uniform machines (rather than unrelated), setup times are sequence-dependent, while also introducing machine eligibility. The heuristic algorithm proposed solves instances of size up to 80 jobs and 20 machines, while for specific cases solutions up to 500 jobs and 20 machines were obtained. On the other hand, Peyro (2020) remove the job splitting property, while adding more types of resource constraints.

Third, we contribute to the design of effective exact methods. Many of the aforementioned works have tried to solve variants of this problem via exact methods. However, due to the problem's computational complexity, efficient solutions come at a cost: either optimal solutions are obtainable for only small size instances or some of the properties are removed to solve optimally a simplified version. The most important factors that increase computational complexity are the job splitting property, which greatly enlarges the solution

space, and the resource constraint, which enforces the modeler to increase the number of variables. Therefore, it seems reasonable to decompose the problem into two formulations. This is a nontrivial task as multiple decompositions are plausible. For example, the first component may deal with assignment and splitting of jobs, while the second one to settle resource allocation; job sequencing per machine might be handled by either. Thus, we present two different Logic-Based Decomposition Methods (LBBD), which are able to solve instances of 200 jobs and 20 machines (for makespan, when coupled with an efficient heuristic) under two different optimization criteria; makespan and tardiness. To exploit the advantages of different optimizing methods, we opt for a LBBD algorithm, which is consisted of a MILP formulation of a master problem and a CP formulation of a subproblem. Two families of optimality cuts will also be presented. The implementation of our algorithm on randomly generated instances proves that the performance gap between exact and heuristic methods can be reduced. We should, also, mention that we manage to solve larger instances than 4 current literature state of the art methods, even while considering a more elaborate problem variation.

Outline. This document is structured as follows: In Section 2, we formally describe the problems studied, while providing novel and effective lower bounds and a three-stage heuristic for the makespan minimization problem. In Section 3, we explain the decomposition methodology along with providing the necessary proofs. In Section 4, we numerically evaluate the algorithms developed on benchmark instances, created similarly to current literature. In Section 5, we conduct experiments based on real datasets provided from PIACENZA. In section 6, we perform experiments which provide important findings on its parameters as business consultation and provide policies for unexpected events. Finally, Section 7 sums up our results and discusses some ideas for further work.

4.2 Problem Description

We consider a set of machines M and a set of jobs J^*. To initialize each machine, we let $J = J^* + \{0\}$, where 0 is a dummy job. Each job $j \in J$ is related to a processing time $p_{j,m}$ per machine $m \in M$ and a setup time $s_{i,j,m}$ that varies with respect to both the machine m and the job $i \in J$ processed just before j in m. At each point in time, up to R machines can be set up.

4.2.1 Problem A: Weaving Scheduling - Makespan

Since our motivation comes from the weaving process in textile manufacturing, we denote “Weaving Scheduling – Makespan” the problem under the PMS environment combined with the aforementioned properties and with the goal of makespan minimization. The problem is NP-hard even in the case where setup times are independent and there are no setup resource constraints (Correa et al. 2015).

4.2.1.1 Lower Bounds of Weaving Scheduling - Makespan

Model Parameters and variables are shown in Tables 4 and 5 respectively.

Model Parameters	
J	The set of jobs - including a dummy job 0
J^*	The set of jobs - excluding the dummy job 0
M	The set of machines
$p_{i,m}$	The processing time of $i \in J$ on $m \in M$
$s_{i,j,m}$	The setup time of $j \in J$ succeeding job $i \in J$ on $m \in M$
$s_0 \in \mathbb{R}$	The setup time of the first job, $\forall m \in M$
R	A setup resource constraint to indicate that, at each time interval, at most R machines can be set up in parallel
δ	A positive constant for (MIP 1), with $\delta \to 0$

Table 4: Model Parameters

Variables	
$z_{i,m}$	1 if $i \in J^*$ is the first job to be assigned on machine $m \in M$, 0 otherwise
$f_{i,m}$	the percentage of the first job $i \in J^*$ to be processed on machine $m \in M$
$y_{i,m}$	1 if $i \in J^*$ is assigned on machine $m \in M$, 0 otherwise
$x_{i,j,m}$	1 if $j \in J$ succeeds $i \in J^*$ on machine $m \in M$, 0 otherwise
$w_{i,m}$	the percentage of job $i \in J^*$ to be processed on machine $m \in M$
$W_{i,m} \in \mathbb{Z}^+$	the integer percentage of job $i \in J^*$ to be processed on machine $m \in M$
$n_{i,m} \in \mathbb{Z}^+$	auxiliary integer variables that ensure the feasibility of sequencing
$C_{\max} \in \mathbb{R}^+$	the makespan of the schedule
$\hat{\mu}_{i,m}$	interval variables that indicate the time interval of setup of job $i \in J^*$ on machine $m \in M$

Table 5: Decision Variables

We formulate two mixed integer program formulations (MIP 1) and (MIP 2). We define (MIP 1), which distinguishes jobs between the ones that are firstly assigned - assuming that they require an a priori constant setup time s_0 - and the ones that follow. Constraints (1) ensure that if a part of a job $i \in J^*$ is processed on a machine $m \in M$, then it should be also assigned. Constraints (2) are similar to (1) but for the job assigned first, while (3) ensure that a job cannot be assigned on a machine in both manners. Constraints (4) say that, if a job is assigned as first on a machine, then a percentage of this job will be processed on this machine. Constraints (5) ensure that each job is fully processed. Constraints (6) ensure that at most one part of a job will be assigned as first on each machine. Constraints (7) ensure that we cannot assign a job to a machine without having already assigned a first job to that machine. Constraints (8) calculate the total load of each machine, taking into consideration setup times, and ensures that it is less than or equal to the makespan of the schedule. Note that in cases where the setup time of the first job is equal to 0, we substitute s_0 with $-\delta$ in Constraints (8).

$$
\begin{aligned}
&\text{(MIP 1)}: && C_{\max} && \\
&&& \text{s.t.}: && \\
&&& y_{i,m} \geq w_{i,m} && \forall i \in J^*, m \in M && (1)\\
&&& z_{i,m} \geq f_{i,m} && \forall i \in J^*, m \in M && (2)\\
&&& z_{i,m} + y_{i,m} \leq 1 && \forall i \in J^*, m \in M && (3)\\
&&& z_{i,m} - f_{i,m} \leq 1 - \delta && \forall i \in J^*, m \in M && (4)\\
&&& \sum_{m \in M} w_{i,m} + f_{i,m} = 1 && \forall i \in J^* && (5)\\
&&& \sum_{i \in J^*} z_{i,m} \leq 1 && \forall m \in M && (6)\\
&&& \sum_{i \in J^*} z_{i,m} \geq \delta \sum_{i \in J} y_{i,m} && \forall m \in M && (7)\\
&&& \sum_{j \in J^*} (f_{j,m} \cdot p_{j,m} + s_0 \cdot z_{j,m} + w_{j,m} \cdot p_{j,m} + y_{j,m} \cdot \min_{i \in J^*} s_{i,j,m}) \leq C_{\max} && \forall m \in M && (8)\\
&&& y_{i,m}, z_{i,m} \in \{0,1\}, f_{i,m}, w_{i,m} \in [0,1], C_{\max} \in \mathbb{R}^+, && \forall i,j \in J^*, m \in M &&
\end{aligned}
$$

(MIP 2) is similar to the formulation in Fotakis et al. (2020). It performs splitting and assignment, based only on processing times, and consists of the following constraints with the objective of minimizing C_{max}:

$$\sum_{i \in J^*} (w_{i,m} \cdot p_{i,m}) \leq C_{\max} \qquad \forall m \in M \qquad (9)$$

$$\sum_{m \in M} w_{i,m} = 1 \qquad \forall i \in J^* \qquad (10)$$

Lemma 4.2.1. Consider an instance I of the Weaving Scheduling - Makespan problem. The solutions of (MIP 1) and (MIP 2) are lower bounds to the optimal solution on I.

Proof. The difference between the two formulations is that (MIP 1) distinguishes jobs to the ones which are processed first and the rest, while (MIP 2) considers only processing times. Both formulations define relaxations of Weaving Scheduling, in which resource constraints are neglected and setup times are set to their minimum values and while assuming a constant setup time is common to both formulations and to Weaving Scheduling.

Given Lemma 4.2.1, a lower bound for an instance of the Weaving Scheduling - Makespan problem can be computed by choosing the maximum between the optimal solutions to the two models. Let us also define (MIP 3), which takes into consideration setup times during the assignment step as in (MIP 1), but does not distinguish jobs based on whether they are assigned first. It consists of (1) with the objective of minimizing C_{max} and the following constraints:

$$\sum_{j \in J^*} (w_{j,m} \cdot p_{j,m} + y_{j,m} \cdot \min_{i \in J^*} s_{i,j,m}) \leq C_{\max} \qquad \forall m \in M \qquad (11)$$

$$\sum_{m \in M} w_{i,m} = 1 \qquad \forall i \in J^* \qquad (12)$$

Contrary to the first two formulations, (MIP 3) offers a lower bound only if the (constant) setup time of the first job is the largest among all setup times.

Lemma 4.2.2. Consider an instance I of the Weaving Scheduling problem. Then, the solution of (MIP 3) is a lower bound to the optimal solution on I if and only if $s_0 \geq s_{ijm}$, $\forall m \in M$, $i, j \in J^*$.

Proof. Exactly as (MIP 1) and (MIP 2), (MIP 3) considers that setup times are set to their minimum values and neglects resource constraints. Assuming $s_0 \geq s_{ijm}$, and given that and we take under consideration only the $\min_{i \in J} s_{i,j,m}$, it is clear that the sum of setup times of the obtained solution on I will never be greater than the corresponding sum in the optimal solution. Thus, (MIP 3) may be considered as a lower bound to the optimal solution on I. We show the inverse, i.e., that (MIP 3) no longer provides a lower bound if $s_0 < s_{ijm}$, by means of an example.

For the instance of Table 6, the lower bound calculated by (MIP 3) is 10.5 (the ones by (MIP 1) and (MIP 2) are 9 and 7.5, respectively). It becomes easy to see that the optimal solution is as shown in Figure 7 and has a makespan of 10.

J \ M	1	2
1	4	7
2	5	5
3	5	3
4	3	4

J \ J	1	2	3	4
1	0	1	1	2
2	1	0	2	2
3	2	2	0	1
4	1	1	1	0

J \ J	1	2	3	4
1	0	1	3	3
2	2	0	2	1
3	0	1	0	0
4	0	1	0	0

Table 6: Random Instance for Lemma 4.2.2. Table on the left shows the processing times, while tables on the right show setup times.

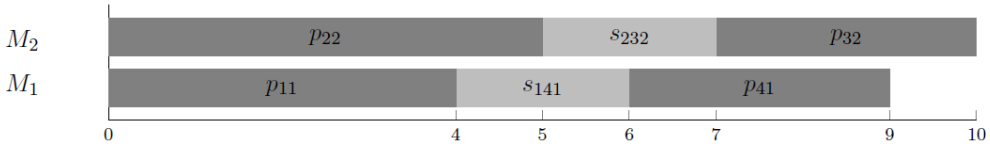

Figure 7: Optimal Solution of Table 6

In terms of computational complexity, it is clear that the proposed lower bounds are derived through mixed integer linear programming formulations, and thus they are NP-hard. Still, even for (MIP 1), which is the most involved formulation, quite large instances of 1000 jobs and 20 machines (with $8 \cdot 10^4$ variables and constraints) were solved in less than 1 minute, while larger ones of 3000 jobs and 20 machines, having $2 \cdot 10^5$ variables and constraints, are solved in less than 2 minutes. Therefore, (MIP 1) and (MIP 2) are useful for obtaining lower bounds, whereas (MIP 3) remains as an alternative model to be used by the primal heuristic discussed in the next section. We propose a three-stage greedy heuristic algorithm (GHA), which solves the problem iteratively and is able to return fast and efficient solutions even for large size instances. The solution of GHA is used both as an upper bound and as an initial solution to the LBBD method described next.

4.2.1.2 A primal greedy heuristic

GHA has three stages. This first stage with splitting of orders to parts and assigning these parts to the machines. The second stage creates the sequences of the job parts on each machine, based on the sequence-dependent setup times. The last one ensures that setup resource constraints are not violated by creating appropriate time windows.

Algorithm 1 GHA: A THREE-STAGE GREEDY HEURISTIC

1: max_assgn $= |J| \cdot |M|$, $C_{\max} \rightarrow \infty$;
2: **while** max_assgn $\geq |J|$ **do**
3: Solve (MIP 1) or (MIP 2) or (MIP 3) and let $\mathcal{Y}$ be the number of job parts assigned over all machines;
4: Set max_assgn $= \mathcal{Y}$;
5: Run SEQUENCING STAGE;
6: Run RESOURCE MANAGEMENT STAGE;
7: Let ld_m be the load of each machine $m \in M$;
8: If necessary update $C_{\max}$ with $\max_{m \in M}\{ld_m\}$
9: Set max_assgn $=$ max_assgn $- 1$
10: **return** $C_{\max}$;

In the Assignment stage, we may choose any among (MIP 1), (MIP 2), (MIP 3) to compute an assignment of job parts over all machines. To capture the iterative solution method of GHA, we include in any formulation used the following:

$$\sum_{i \in J} \sum_{m \in M} (y_{i,m} + z_{i,m}) \leq \max\_assgn \qquad \forall m \in M \qquad (13)$$

$$w_{j,m} \cdot p_{j,m} \geq \min_{i \in J^*} s_{i,j,m} \qquad \forall i, j \in J^*, m \in M \qquad (14)$$

Constraints (13) limit the number of the possible assignments, performed in each iteration. Constraints (14) ensure that assignments which will consume more setup time than processing time are excluded. Especially for (MIP 2), we also need to include Constraints (1). The Sequencing Stage orders the assigned job parts by solving an asymmetric TSP (aTSP) for each machine, where nodes are the job parts and the distance from a node to another equals the sequence-dependent setup time of the associated pair of jobs plus the processing time of the job part of the destination node. The aTSPs are solved to optimality, in fact pretty fast through the method of Roberti et al. (2012). In the Resource Management Stage, 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 whose setup has already been determined. The idea here is that, by starting from the most loaded machine, we reduce significantly the effect of idle intervals between consecutive job executions on the final makespan. Which among the three MIPs is most appropriate regarding GHA can be determined only computationally. Our experience has shown that (MIP 3) offers the most effective balance between solution quality and time.

4.2.2 Problem B: Weaving Scheduling - Tardiness

Now, let us associate each job $j \in J$ with a deadline d_j. We note that the deadline of each job is not strict; if the completion time C_j of job j violates d_j, then a tardiness $T_j = \max\{0, C_j - d_j\}$ is charged in the objective value. Additionally, even though we obtain all the properties of “Weaving Scheduling – Makespan”, we decide to ignore job splitting. Thus, on a similar manner to 4.2.1., we denote “Weaving Scheduling – Tardiness” as the PMS problem combined with the aforementioned properties but under the goal of total tardiness minimization.

4.3 An exact method Benders Decomposition

Benders (1962) defines an interative method where master problem M is strengthened by a set of cuts at each iteration that solves a subproblem S. As classical Benders decomposition is valid for non-integer subproblems, Hooker et al. (2003) presented the extended Logic-Based Benders Decomposition (referred as LBBD hereinafter). Let us briefly describe LBBD, while also introducing our notation. Let $P = \min\{f(x) + g(y) | x \in D_x, y \in D_y\}$ be the formulation of a combinatorial optimization problem, in which x, y are groups of variables and D_x, D_y are their respective domains. f(x) and g(y) are linear cost functions. P is decomposed into the master problem $M = \min \{z | z \geq f(x), x \in D_x\}$ and the subproblem $S = \min\{g(y) + f(x) | y \in D_y\}$, in which z is an upper bound of f(x) and $\hat{x}$ is a feasible solution of M. For iteration k, M_{k-1} provides a feasible solution $\hat{x}_{k-1}$ to S_k. If the objective value of S_k is equal with the objective value of M_{k-1}, then convergence is reached and the optimal solution is found.

Otherwise, a set of inequalities, called cuts are added to M_k:

$$z \geq \beta^k(x) \qquad \forall k = 1, ..., K$$

in which $\beta^k(x)$ is a bounding function that provides a lower bound of z. If $x = \hat{x}_{k-1}$, then $\beta^k(x)$ will be equal with the objective value of S_k. These cuts are called optimality cuts, as they ensure that their respective solution will not be computed again, unless it is the actual optimal one. If S_k is infeasible for a solution $\hat{x}_{k-1}$, a set of feasibility cuts restricts M to compute $\hat{x}_{k-1}$ in any succeeding iteration. As no infeasibilities may occur in our case, we will focus on optimality cuts only.

4.3.1 LBBD for Weaving Scheduling - Makespan

Now, that we have defined the basic elements of LBBD, let us move on to the formulation for the Weaving Scheduling – Makespan.

4.3.1.1 The master problem M for Weaving Scheduling - Makespan

Related decompositions for scheduling problems avoid overloading the master problem, as the aTSP Problem is too complex to be solved (near) optimally. Thus, our master problem computes the optimal assignments and sequences of jobs to machines, thus exploiting also the solution offered by GHA. The master problem assumes an infinite number of workers, i.e., unlimited setups at each period. To define a finite domain of variables, we convert the fractional variables that indicate the proportion of splits into integer values of percentage. Hence, integer variables $W_{i,m} \in \{0, 100\}$ indicate the percentage of job i that is assigned to machine m, rounded to the closest integer value $W_{i,m} = w_{i,m} \cdot 100$. If $W_{i,m}$ were regular fractional variables that lay in [0, 1], the decomposition algorithm would never converge, as the domain of variables would not be finite. The above do not suffice for an effective decomposition, as subtours may occur. Typically, the constraints that define completion times also ensure the elimination of subtours. However, as the computation of completion times is not compulsory for a makespan objective, we opt for the subtour elimination constraints of Miller-Tucker-Zemlin (Miller et al. 1960). Hence, we define auxiliary variables $n_{i,m}$, indicating the order of job i to machine m. The formulation of the master problem in iteration $k - 1$ is M_{k-1}.

$\mathcal{M}_{k-1}$:

min z

subject to:

Assignment of jobs to machines

$$\sum_{m \in M} W_{i,m}^{k-1} = 100 \qquad \forall i \in J^* \qquad (15)$$

$$100 \cdot y_{i,m}^{k-1} \geq W_{i,m}^{k-1} \qquad \forall i \in J^*, m \in M \qquad (16)$$

Sequencing of jobs

$$y_{i,m}^{k-1} = \sum_{j \in J, j \neq i} x_{i,j,m}^{k-1} \qquad \forall i \in J^*, m \in M \qquad (17)$$

$$y_{i,m}^{k-1} = \sum_{j \in J, j \neq i} x_{j,i,m}^{k-1} \qquad \forall i \in J^*, m \in M \qquad (18)$$

$$\sum_{j \in J} x_{0,j,m}^{k-1} \leq 1 \qquad \forall m \in M \qquad (19)$$

$$n_{i,m}^{k-1} - n_{j,m}^{k-1} + |J| \cdot x_{i,j,m}^{k-1} \leq |J| - 1 \qquad \forall i, j \in J^*, m \in M \qquad (20)$$

$$n_{i,m}^{k-1} \leq |J| - 1 \qquad \forall i \in J^*, m \in M \qquad (21)$$

Objective function

$$z \geq \sum_{j \in J^*} \frac{p_{j,m}}{100} \cdot W_{j,m}^{k-1} + \sum_{i \in J} \sum_{j \in J^*} s_{i,j,m} \cdot x_{i,j,m}^{k-1} \qquad \forall m \in M \qquad (22)$$

Variables

$$x_{i,j,m}^{k-1} \in \{0, 1\} \qquad \forall i \in J, j \in J, m \in M$$

$$y_{i,m}^{k-1} \in \{0, 1\} \qquad \forall i \in J, m \in M$$

$$n_{i,m}^{k-1}, W_{i,m}^{k-1} \in \mathbb{Z}_+ \qquad \forall i \in J, m \in M$$

Let us explain our constraints in more detail. Constraints (22) define the makespan as the greatest sum of processing and setup times of all machines. Constraints (15) ensure that all jobs will be entirely assigned to machines, while (16) ensure that all jobs will be assigned to machines. Moving now to sequencing, constraints (17) and (18) ensure that if job i is assigned to machine m, it will also be assigned to one (different) preceding and a succeeding job. Constraints (19) extend (17) for the dummy job 0. Constraints (20) eliminate subtours, ensuring that the order of each job will be smaller than its succeeding one. The upper bound (21) on variables $n^{k-1}_{i,m}$ is the number of jobs . Variables $x^{k-1}_{i,j,m}$ are equal to 1 if job j succeeds job i in machine m, 0 otherwise. $y^{k-1}_{i,m}$ are binary variables that indicate whether job i is assigned to machine m and the respective variables $W^{k-1}_{i,m}$ indicate the integer percentage of job i that is assigned to m. Auxiliary variables $n^{k-1}_{i,m}$ denote the order of job i in the sequence of machine m. Let $\hat{x}^{k-1} = \{\hat{x}^{k-1}_{i,j,m}, \hat{y}^{k-1}_{i,m}, \hat{W}^{k-1}_{i,m}\}$ be the optimal solution and $\hat{z}^{k-1}$ be the objective value of M_{k-1} after iteration $k-1$. Variables $\hat{n}^{k-1}_{i,m}$ are auxiliary, therefore they do not contribute to the input of $\hat{x}_{k-1}$ to the subproblem. As already mentioned, the LBBD hardly solves instances of more than 10 jobs, because the master problem M fails to compute feasible solutions for larger datasets. As we have already computed a primal solution by the GHA heuristic, we can supply it to the solver as a warm start (CPLEX). In particular, as the master problem does not include the Resource Management Stage of the problem, we use the solution of the Assignment Stage and the Sequencing Stage of the GHA heuristic as a primal feasible solution of M_0 for the first iteration of the LBBD algorithm. Combining the GHA heuristic with the master problem increases the capabilities of our exact method, so that large instances can be solved, as we will show next.

4.3.1.2 The subproblem S for Weaving Scheduling - Makespan

As all assignments and sequences are computed by M, the subproblem ensures that the setups will not simultaneously occupy more workers than the available ones. We also consider a constraint that imposes a daily setup time limit u (intended to be used for the real datasets only). As it is proved from experience that CP formulations handle these kinds of constraints more efficiently than equivalent MILP formulations would do, we opt for a Constraint Programming formulation. The set of used machines is denoted by $\bar{M}_{k-1} = \{m \in M | \Sigma_{i \in J^*} \hat{y}^{k-1}_{i,m} \geq 1\}$ and the fixed sequences of jobs on machines are denoted by $\bar{v}^{k-1}_m$. Let $\bar{s}^{k-1}_{i,m}$ be the setup time and $\bar{p}^{k-1}_{i,m}$ be the processing time of the i th job of sequence $\bar{v}^{k-1}_m$. Let S_k be the formulation of the subproblem in iteration k.

Interval variables $\bar{\mu}^k_{i,m}$ indicate the time interval of the setup of the i th job of sequence $\bar{v}^{k-1}_m$. The size of the interval is the setup time that has been computed by the solution of M_{k-1}.

$\mathcal{S}_k$:

$$\min \zeta^k$$

subject to:

$$\texttt{Cumulative}((\texttt{start\_of}(\bar{\mu}^k_{i,m})), (\bar{s}^{k-1}_{i,m}), 1, R) \tag{23}$$

$$\texttt{start\_of}(\bar{\mu}^k_{i,m}) \geq \texttt{end\_of}(\bar{\mu}^k_{i-1,m}) + \bar{p}^{k-1}_{i-1,m} \quad \forall m \in \bar{M}_{k-1}, i = 2, ..., |\bar{\nu}^{k-1}_m| \tag{24}$$

$$\zeta^k \geq \texttt{end\_of}(\bar{\mu}^k_{i,m}) + \bar{p}^{k-1}_{i,m} \quad \forall m \in \bar{M}, i = 1, ..., |\bar{\nu}^{k-1}_m| \tag{25}$$

$$\bar{\mu}^k_{i,m} : \texttt{interval}(\text{size} = \bar{s}^{k-1}_{i,m}) \quad \forall m \in \bar{M}, i = 1, ..., |\bar{\nu}^{k-1}_m|$$

Constraint (23) is a global constraint, ensuring that the number of simultaneous setups that start at ***start_of***($\bar{\mu}^k_{im}$), terminate after $\bar{s}^{k-1}_{i,m}$ and occupy 1 worker cannot exceed the number of available workers R. Constraints (24) ensure that the processing and setup times of all jobs will not overlap in the same sequence. Constraint (25) defines the objective function ζ^k, which is the updated value of makespan. Let ζ^k be the objective value of S_k. If $\hat{z}^{k-1} < \zeta^k$, a set of optimality cuts will be added to M_k. Otherwise, ζ^k is the minimum makespan of the problem.

4.3.1.3 Optimality cuts and algorithm

This is the most technical part of the LBBD exposition. Let P be the original optimization problem. The LBBD gives rise to Algorithm 2, which converges to the optimal solution of P if the proposed optimality cuts (29)-(32) are valid.

Algorithm 2 A Logic-Based Benders Decomposition Algorithm

1: Let $k = 0$ be an iteration number, CONVERGENCE = `False`, K be the maximum number of iterations and E be the maximum gap of convergence;
2: Solve $\mathcal{M}_0$, using the solution of GHA as a warm start, and let $\hat{\mathrm{X}}^k = \{\hat{x}^k_{ijm}, \hat{y}^k_{im}, \hat{W}^k_{im}\}$ be its optimal solution and $\hat{z}^k$ its objective value;
3: Let z be the LOWER BOUND of $\mathcal{P}$ and $\hat{z}^k$ its incumbent value;
4: **while** CONVERGENCE = `False` **do**
5: Set $k = k + 1$
6: Solve $\mathcal{S}_k$;
7: Let ζ^k be the objective value of $\mathcal{S}_k$ and $\epsilon = \frac{\zeta^k - z}{z}$ be the gap;
8: **if** $\zeta^k = min\{\zeta^r | r = 1, ..., k\}$ **then**
9: Let UPPER BOUND = ζ^k and $\bar{\mathrm{X}} = \hat{\mathrm{X}}^{k-1}$ be the incumbent optimal solution;
10: **if** $\epsilon > E$ **and** $k < K$ **then**
11: Add (29)-(32) to $\mathcal{M}_k$;
12: Solve $\mathcal{M}_k$;
13: Let $\hat{z}^k$ be equal to the objective value of the solution $\hat{\mathrm{X}}^k$ of $\mathcal{M}_k$ and $z = \hat{z}^k$;
14: **else**
15: CONVERGENCE = `True`;
16: return UPPER BOUND and $\bar{\mathrm{X}}$;

CONVERGENCE	Binary parameter, indicating whether convergence is reached or not
K	Maximum number of iterations; after K iterations, the algorithm terminates
E	Maximum gap of convergence; if the gap is less than E, the algorithm terminates
LOWER BOUND	The tightest lower bound of $\mathcal{P}$
UPPER BOUND	The tightest upper bound of $\mathcal{P}$

We construct a bounding function $\beta^k(x_k)$ of variables x_k in iteration k which adheres to the following conditions:

C1: the bounding function provides a valid lower bound of z, that is, $z \geq \beta^k(x_k)$ for all feasible solutions x_k.

C2: if $x_k = \hat{x}_{k-1}$, then $z = \beta_k(\hat{x}_{k-1})$, for which $\hat{x}_{k-1}$ is the solution of M_{k-1} for iteration $k - 1$.

Let P^{k-1} be the set of assigned pairs of jobs to machines after iteration $k - 1$: $P^{k-1} = \{p = (i, m) | \hat{y}^{k-1}_{i,m} = 1 \; \forall \; i \in J^*, m \in M\}$. Now, we construct a set A^{k-1} which includes all assignments (i, j, m) for which $\hat{x}^{k-1}_{i,j,m} = 1$: $A^{k-1} = \{a = (i, j, m) | \hat{x}^{k-1}_{i,j,m} = 1\}$. Let $\beta^k(x_k)$ be the bounding function of iteration k:

$$\beta^k(\mathrm{x}_k) = \begin{cases} \zeta^k & \text{if } \{a \in A^{k-1}, p \in P^{k-1} | x^k_a = 0, W^k_p \neq \hat{W}^{k-1}_p\} = \varnothing \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

for which, if all assignments $a \in A^{k-1}$ and splits of pairs $p \in P^{k-1}$ are the same for iterations $k - 1$ and k, then the bounding function is equal with the objective value of the subproblem S_k. On the contrary, if any assignment $a \in A^{k-1}$ or split of pair $p \in P^{k-1}$ differs in k, then the value of the bounding function is 0. Now, we construct a set of linear expressions, so that a cut of the following form: $z \geq \beta^k(x_k)$ is added to M_k. We define binary variables ρ^k_p which are equal to 1 if $W^k_p = W^{k-1}_p$, $p = (i, m) \in P$, 0 otherwise. The following conjunction is trivial

$$W^k_p = \hat{W}^{k-1}_p \longleftrightarrow W^k_p \geq \hat{W}^{k-1}_p \wedge W^k_p \leq \hat{W}^{r-1}_p. \tag{27}$$

Let $\lambda_p^{\geq k}$ and $\lambda_p^{\leq k}$ be binary variables that are equal to 1, if $W^k_p \geq \hat{W}^{k-1}_p$ and $\hat{W}^k_p \leq W^{k-1}_p$, respectively, and 0 otherwise. Such binary indicator variables are known as literals (Lam et al. 2020). By (27):

$$\lambda_p^{\geq k} = 1 \wedge \lambda_p^{\leq k} = 1 \longrightarrow \rho_p^k = 1 \tag{28}$$

To construct the linear expression of (27), we convert the disjunctive indicator constraints of (Lam et al. (2020), inequality (6)) to the following conjunctive form:

$$\hat{W}_p^{k-1} - W_p^k + v \leq V \cdot \lambda_p^{\leq k} \qquad \forall p = (i,m) \in P^{k-1} \tag{29}$$

$$W_p^k - \hat{W}_p^{k-1} + v \leq V \cdot \lambda_p^{\geq k} \qquad \forall p = (i,m) \in P^{k-1} \tag{30}$$

where υ is a small number ($0 < \upsilon < 1$) and V is a big one ($V \geq 100 + \upsilon$). If both of (29) and (30) hold for a pair $p \in P^{k-1}$, then ρ^k_p will be equal to 1. (28) is equivalent to

$$\rho_p^k + 1 \geq \lambda_p^{\geq k} + \lambda_p^{\leq k} \qquad \forall p = (i,m) \in P^{k-1}. \tag{31}$$

The following cut ensures that if all assignments $x^k_{ijm} = 1$ for $a = (i, j, m) \in A^{k-1}$ and all splits $W^k_p = \hat{W}^{k-1}_p$ for $p = (i, m) \in P^{k-1}$, then ζ^k will be a lower bound of z.

$$z \geq \zeta^k - \zeta^k \cdot \left[(|A^{k-1}| - \sum_{a \in A^{k-1}} x_a^k) + (|P^{k-1}| - \sum_{p \in P^{k-1}} \rho_p^k) \right] \tag{32}$$

We can now show the following, in a style similar to (Hooker (2007), Theorem 1).

Theorem 3.1. *If $\beta^k(\mathrm{x}_k)$ adheres to conditions C_1 and C_2 and the domain D_Υ of variables $\Upsilon = \{\bar{\mu}^k_{i,m}\}$ is finite, cuts (29)-(32) are valid, hence Algorithm 2 converges to the optimal solution of $\mathcal{P}$ after finitely many steps (for $K = +\infty$ and $E = 0$).*

Proof. Let $f(\mathrm{x}_k)$ be the objective function z of $\mathcal{M}_k$ in iteration k. We must prove that $f(\bar{\mathrm{x}}_k) \geq \beta^k(\bar{\mathrm{x}}_k)$ for all feasible solutions $\bar{\mathrm{x}}_k$ (C_1), and, in particular, $f(\bar{\mathrm{x}}_k) = \beta^k(\bar{\mathrm{x}}_k)$ for $\bar{\mathrm{x}}_k = \hat{\mathrm{x}}_{k-1}$ (C_2).
Let $\bar{\mathrm{x}}_k$ be a feasible solution of $\mathcal{M}_k$. If $\bar{\mathrm{x}}_k = \hat{\mathrm{x}}_{k-1}$, then $\bar{x}^k_a = \hat{x}^{k-1}_a = 1$ for all assignments $a \in A^{k-1}$ and $\bar{W}^k_p = \hat{W}^{k-1}_p$ for all pairs $p \in P^{k-1}$, hence, by (26), $\beta^k(\bar{\mathrm{x}}_k) = \zeta^k$. By (32), $z = \zeta^k \rightarrow z = \beta^k(\bar{\mathrm{x}}_k) \rightarrow f(\bar{\mathrm{x}}_k) = \beta^k(\bar{\mathrm{x}}_k)$. If $\bar{\mathrm{x}}_k \neq \hat{\mathrm{x}}_{k-1}$, there is at least one assignment $a \in A^{k-1}$ for which $\bar{x}^k_a = 0$ or one split of a pair $p \in P^{k-1}$ for which $\bar{W}^k_p \neq \hat{W}^{k-1}_p$. By (26), $\beta^k(\bar{\mathrm{x}}_k) = 0$, which is always a lower bound of $z = f(\bar{\mathrm{x}}_k)$.
As C_1 and C_2 hold and the domain of variables $\bar{\mu}^k_{im}$ is finite, Theorem 3.1 is valid. □

4.3.2. LBBD for Weaving Scheduling – Tardiness

On a similar way as for Weaving Scheduling – Makespan, we move on to describe the LBBD formulation for Weaving Scheduling – Tardiness.

4.3.2.1. The Master Problem for Weaving Scheduling – Tardiness

The proposed formulation of M is a MILP model, which provides tight lower bounds for all tardiness variables. Variables Y_{im} are equal to 1 if job i ∈ J is assigned to machine m ∈ M, 0 otherwise. X_{ijm} is equal to 1 if job i ∈ J is assigned to slot j ∈ J of machine m ∈ M, or 0 otherwise. C_{jm} is the completion time of slot j ∈ J in machine m ∈ M, t^d_{jm} is the due time of slot j in machine m, and T_{jm} is the tardiness of slot j in m. To avoid big-M constraints, we omit the sequence dependency of setup times, by replacing the original s_{jim} values with $\bar{s}_{im} = \min_{j \in J} s_{jim}$, s_{jim} being the setup time of job i in machine m, if the precedent job is j. As $\bar{s}_{im} \leq s_{jim}$ ∀i, j ∈ J, m ∈ M, the optimal objective value of M is a lower bound of the global optimal objective value of P.

$\mathcal{M}$:

$$\min z \geq \sum_{j \in J} \sum_{m \in M} T_{jm} \tag{33}$$

subject to:

$$\sum_{m \in M} Y_{im} = 1 \qquad \forall i \in J \tag{34}$$

$$\sum_{j \in J} X_{ijm} = Y_{im} \qquad \forall i \in J, m \in M \tag{35}$$

$$\sum_{i \in J} X_{ijm} \leq 1 \qquad \forall j \in J, m \in M \tag{36}$$

$$C_{0m} = \sum_{i \in J} (p_{im} + \bar{s}_{im}) \cdot X_{i0m} \qquad \forall j \in J, m \in M \tag{37}$$

$$C_{jm} = C_{j-1,m} + \sum_{i \in J} (p_{im} + \bar{s}_{im}) \cdot X_{ijm} \qquad \forall j \in J, m \in M \tag{38}$$

$$t^d_{jm} = \sum_{i \in J} d_i \cdot X_{ijm} \qquad \forall j \in J, m \in M \tag{39}$$

$$T_{jm} \geq C_{jm} - t^d_{jm} \qquad \forall j \in J, m \in M \tag{40}$$

$$Y_{im} \in \{0,1\} \qquad \forall i \in J, m \in M$$

$$X_{ijm} \in \{0,1\} \qquad \forall i \in J, j \in J, m \in M$$

$$C_{jm}, t^d_{jm}, T_{jm} \geq 0 \qquad \forall j \in J, m \in M$$

Constraints (34) and (35) ensure that each job will be assigned to one slot of one machine. Constraints (36) ensure that each slot j ∈ J can process only one job i ∈ J at most. Constraints (37) initialize the completion times of machine m, and Constraints (38) define the completion time of slot j as the completion time of the preceding slot j−1, added by the processing and setup times of the job i that is assigned to j. Constraints (39) define the due time of slot j, according to the due time of the assigned job i, as Constraints (40) compute the tardiness of slot j. The objective function (33) is an upper bound of the sum of all tardiness values. The solution of M is a set of sequences $\bar{M}$, in which each sequence m ∈ $\bar{M}$ is associated with machine m ∈ M. The slots of the sequences are assigned to the respective job for which X_{ijm} = 1 (i.e., m_j = i indicates that the slot j of sequence m ∈ $\bar{M}$ is connected with job i ∈ J).

4.3.2.2. The subproblem for Weaving Scheduling – Tardiness

One of the most significant benefits of decomposition methods is the combination of different optimizing methods. As several recent studies prove that Constraint Programming

formulations are more efficient for scheduling problems that involve sequence-dependencies and resource constraints, we opt for a CP model, denoted by S. The solution of S considers the actual setup times s_{ijm} of the sequences of jobs that are supplied by the solution of M, as well as it ensures that the optimal schedule will respect the availability of workers R, which was neglected by M. We define a set of interval variables σ_{mj} for each slot j of sequence m ∈ $\bar{M}$.

$\mathcal{S}$:

$$\min \zeta \geq \sum_{i \in J} T_i^* \tag{41}$$

subject to:

$$\texttt{Cumulative}(\texttt{startOf}(\sigma_{mj}), \texttt{sizeOf}(\sigma_{mj}), 1, R) \tag{42}$$

$$C_{mj}^* = \texttt{endOf}(\sigma_{mj}) + p_{m_j m} \qquad \forall m \in \bar{M}, j \in m \tag{43}$$

$$T_{m_j}^* \geq C_{m_j}^* - d_{m_j} \qquad \forall m \in \bar{M}, j \in m \tag{44}$$

$$\texttt{startOf}(\sigma_{mj}) \geq C_{mj-1}^* \qquad \forall m \in \bar{M}, j \in m \tag{45}$$

$$\texttt{sizeOf}(\sigma_{mj}) = s_{m_{j-1} m_j m} \qquad \forall m \in \bar{M}, j > 0 \in m \tag{46}$$

$$\texttt{sizeOf}(\sigma_{m0}) = s_{m_0 m}^0 \qquad \forall m \in \bar{M} \tag{47}$$

$$\sigma_{mj} : \texttt{Interval} \qquad \forall m \in \bar{M}, j \in m$$

$$C_{mj}^* \geq 0 \qquad \forall m \in \bar{M}, j \in m$$

$$T_i^* \geq 0 \qquad \forall i \in J$$

Constraints (46) and (47) define the duration of each setup task as the sequence-dependent setup time of the respective job m_j. The global constraint Cumulative ensures that no more than R setup tasks that start at ***startOf***(σ_{mj}) and occupy 1 worker for ***sizeOf***(σ_{mj}) time, will be executed simultaneously (42). Constraints (43) and (44) compute the updated values of completion times and tardiness respectively. Constraints (45) ensure that each setup task must start later than the completion time of the preceding slot. The objective function ζ (41) is the sum of tardiness.

4.3.2.3. Optimality Cuts for Weaving Scheduling – Tardiness

To secure the convergence of the LBBD to the optimal solution of P, a set of valid optimality cuts must be added to M, after a new upper bound is obtained. We can safely assume that, if M computes a set of sequences $\bar{M}$ that has been already fed to S before, implying an upper bound equal to U, then the new upper bound is expected to be equal to U as well. Hence, the first family of cuts involves a cumulative optimality cut, which resembles the corresponding cuts of (Hooker, 2007):

$$z \geq U_k - U_k \cdot (|A_{k-1}| - \sum_{a \in A_{k-1}} X_{a_0 a_1 a_2}) \qquad \forall k \tag{48}$$

Let A^{k-1} be the set of assignments of jobs-to-slots-to-machines that have been computed in iteration k −1 (i.e., for each a ∈ A^{k-1}, a_0 = i, a_1 = j, a_2 = m for which X_{ijm} = 1). The assignments of A^{k-1} implied an upper bound U_k. For each iteration k, cut (48) ensures that, if all assignments A^{k-1} are selected again, then the objective value z of M will be greater than the respective upper bound U_k. In other words, M is forced to select a new set of sequences $\bar{M}$ to avoid the objective value U_k, unless the latter one is the minimum feasible. In this case, the lower and upper bounds converge and the optimal solution is found. By modifying analogous proofs, it can be shown that Algorithm 3 converges to the optimal solution of P after finitely many iterations.

Algorithm 3 LBBD Algorithm

Let $k = 0$ be the iteration, K be the maximum number of iterations and E be the maximum gap of convergence;
Solve $\mathcal{M}$ and let z be the lower bound, $\bar{M}$ the solution;
while $k < K$ or GAP $> E$ **do**
    Set $k = k + 1$
    Solve $\mathcal{S}$ for sequences $\bar{M}$;
    Let ζ be the upper bound and GAP $= \frac{\zeta - z}{\zeta}$;
    if GAP $> E$ **and** $k < K$ **then**
        Add (48) to $\mathcal{M}$;
        Solve $\mathcal{M}$;
        Let z be the new lower bound, $\bar{M}$ the new sequences;
    else
        Terminate;
return $\bar{M}$ and z;

4.4 Benchmarking on random datasets

We, now, move on to examine our proposed methods on benchmark datasets.

4.4.1. Benchmark experiments for Weaving Scheduling - Makespan

We randomly generate instances with number of machines |M| ∈ {2, 5, 10, 15, 20} and number of jobs |J| ∈ {10, 20, 30, 40, 50, 80, 100, 150, 200, 300, 400, 500, 700, 1000}, i.e., 70 combinations. To better capture the fact that machines are unrelated, the processing time p_{im} of job i ∈ J in machine m ∈ M is set to $b_i \cdot a_{im}$ plus some noise selected uniformly at random (u.a.r.) from [0, 10], where bi and a_{im} are selected u.a.r. from [1, 10] (as in [Fotakis et al. (2016)). For the sequence- and machine-dependent setup times, we set $s_{ijm} = \alpha_{ijm} \cdot p_{jm}$, where α_{ijm} is selected u.a.r. either from [0.01, 0.1] or from [0.1, 0.2] or from [0.1, 0.5] (as in Kim et al. 2021)). This is a total of 70 × 3 = 210 instances, each solved for 3 different number of working resources R ∈ {1, 3, 5}.

Experiments for the GHA heuristic has been performed on a server with 4 Intel(R) Xeon(R) E2126G @ 3.30GHz processors and 11 GB RAM, running CentOS/Linux 7.0. We used Python 3.8.5 for scripting and Pyomo 5.7.3 with the Gurobi Optimizer 9.1.5 (GUROBI) for solving the MIPs. The LBBD experimentation has used a server with 8 Intel(R) Core(TM) i7-4790 CPU @3.60GHz processors, also running CentOS/Linux 7.0. The solver of the LBBD was CPLEX 20.1 [IBM], via the Python API for the master problem and the CP Optimizer of the DOcplex module for the subproblem.

4.4.1.1 Performance of the GHA heuristic

For the assignment step of GHA, the solver is interrupted after either 60 seconds or if 0.1% optimality gap is reached. For instances with 700 or 1000 jobs, the gap limit is increased to 0.5%. Similar limits are imposed for the aTSP determining the sequencing per machine. Since $s_0 = 0 < s_{ijm}$ for any i, j ∈ J*, m ∈ M, we conclude that we cannot use the solution of (MIP 3) as a lower bound. Table 4.4.1 shows that (MIP 1) offers better lower bounds than (MIP 2) (recall Section 4.2.2.1), where Diff is the % increase in the lower bound computed via (MIP 1) compared to that of (MIP 2). We conclude through Table 7 that (MIP 1) provides, on average, 3.24% tighter lower 15 bounds although requiring more time.

Instance		MIP 1 Time (s)	MIP 2 Time (s)	Diff(%)
J	10	3	< 1	4.38
	20	13	< 1	4.52
	30	19	< 1	4.2
	40	16	< 1	3.9
	50	20	< 1	3.88
	80	15	< 1	3.53
	100	12	< 1	3.43
	150	6	< 1	3.1
	200	3	< 1	2.93
	300	5	1	2.67
	400	7	1	2.5
	500	10	2	2.36
	700	17	5	2.2
	1000	32	4	2.09
M	2	< 1	< 1	2.25
	5	2.3	< 1	3.13
	10	8.3	< 1	3.54
	15	23	2	3.68
	20	27	2	3.69
Alpha	[0.01, 0.1]	5	1	0.7
	[0.1, 0.2]	18	1	3.7
	[0.1,0.5]	16	1	5.32
Mean		13	1	3.24

Table 7: Experiments on LB

Instance		Machines 2 Gap	2 Time	5 Gap	5 Time	10 Gap	10 Time	15 Gap	15 Time	20 Gap	20 Time	Mean Gap	Mean Time
J	10	8.48	< 1	13.26	< 1	12.19	3	18.69	8	15.84	18	13.69	6
	20	7.19	< 1	10.06	1	21.97	52	24	73	33.52	136	19.35	52
	30	6.69	< 1	13.34	2	31.06	150	34.36	137	42.74	347	25.64	127
	40	5.64	< 1	10.8	2	25.13	107	38.23	141	51.56	272	26.27	104
	50	5.98	< 1	10.27	3	22.41	148	41.08	26	57.16	270	27.38	90
	80	6.07	2	7.51	3	23.51	10	41.29	29	59.57	164	27.59	42
	100	6.03	3	7.27	4	19.49	14	40.19	25	55.24	57	25.64	21
	150	5.95	6	7.52	8	17.97	22	33.57	43	51.94	78	23.39	31
	200	5.77	12	7.58	16	16.29	24	34.01	60	50.49	108	22.83	44
	300	5.81	47	8.1	33	14.97	79	30	95	42.57	161	20.29	83
	400	6.03	106	7.49	77	13.63	147	28.91	217	41.75	329	19.56	175
	500	6.1	210	7.54	158	13.38	191	29.25	330	42.86	482	19.83	274
	700	16.28	428	7.58	355	12.74	455	29.04	520	39.79	639	21.09	479
	1000	-	-	7.66	610	13.82	849	29.06	868	39.16	1152	22.43	870
α	[0.01,0.1]	1.05	59	2.24	81	3.62	165	4.39	179	6.83	307	3.63	161
	[0.1,0.2]	8.84	62	10.94	87	20.62	168	36.54	197	48.96	288	25.18	158
	[0.1,0.5]	11.35	67	13.81	105	31.16	150	55.86	175	77.96	307	38.03	161
R	1	7.85	63	13.21	91	38.28	161	72.8	184	99.1	301	46.25	160
	3	6.69	63	6.87	91	8.74	160	13.22	184	21.75	300	11.46	160
	5	6.69	63	6.87	91	8.39	161	10.78	184	12.91	301	9.14	160
Mean		7.08	63	8.98	91	18.47	161	32.27	184	44.59	301	22.28	160

Table 8: Benchmark Experiments of GHA

Further testing, not listed here for brevity, has shown that (MIP 3) is more suitable for the assignment step of GHA as it finds better solutions than (MIP 2) in much less time than (MIP 1). Therefore, Table 8 shows the results for GHA when using (MIP 3) for the assignment step but with the lower bound LB calculated as the maximum between the optima of (MIP 1) and (MIP 2). The running time is measured in seconds and $\text{Gap} = \frac{(\text{GHA}-\text{LB})\cdot 100}{\text{LB}}$. The presentation style follows that of [Kim et al. 2021)]. Instances having 2 machines and 1000 jobs are not solved, as the aTSP exceeds our time limit. The average gap is 22.28% and the average time is 160 seconds. These average values would be significantly lower if excluding the (rather extreme) case of R = 1. For R = 1, large gaps are to be expected, especially when the number of machines increases. However, for R = 3, we see that the average gap drops to 11.46% and for R = 5 to 9.14%. It is worth noting that GHA solves instances with more than 400 jobs and more than 10 machines, i.e., large-scaled regarding current literature. When α ∈ [0.01, 0.1], GHA provides almost optimal solutions at a 3.63% optimality gap relatively fast (161 s in average). As expected, when α increases thus representing higher sequence-dependent volatility, we obtain much higher gaps. This, however, is also attributed to the worse LB, i.e., as once α goes up and the makespan becomes more affected by setup times, (MIP 2) performs worse on assignment and splitting. In addition, we use setup working resources R for longer intervals, thus their availability falls and machines are idling.

4.4.1.2. Performance of LBBD coupled with GHA

Regarding LBBD, K is set to 20 and E is set to 1%, (i.e., Algorithm 2 terminates after 20 iterations or after Gap falls below 1%). We also impose that the algorithm terminates if 10

consecutive iterations do not provide an improved Upper Bound. A time limit of 600 seconds is imposed on the solution of the master problem. If this limit is reached (i.e., the solution of M remains sub-optimal), the Lower Bound of the respective iteration is not necessarily the highest, thus we take the maximum over all iterations to ensure that Gap takes the real optimality gap of P. There is no time limit for the subproblem, as it is solved in a few seconds. As already mentioned, to increase the capabilities of the LBBD, we use the solution of the first two stages of GHA as a warm start for the MILP solver, to provide a primal feasible solution for the master problem in the first iteration of Algorithm 2. In that manner, we exploit the complementary strengths of GHA and LBBD: GHA gives a quick upper bound, which LBBD may or may not improve, and LBBD improves the lower bound thus aiming at a considerably smaller gap. Table 9 shows that this idea is beneficial. Gap equals $\frac{Upper\ Bound - Lower\ Bound}{Lower\ Bound}$ and Time is the convergence time (in seconds). As the Lower Bound of the LBBD is always tighter than the respective LB of Algorithm 2, we also add column GHA, which is the actual optimality gap of the heuristic solutions $\frac{GHA - Lower\ Bound}{Lower\ Bound}$. Although LBBD computes better solutions than GHA, we notice that no such solutions are provided for several datasets with more than 300 jobs. Datasets of 400 and 500 jobs can be handled only for 2 machines and the datasets of 700 and 1000 jobs cannot be solved at all due to memory limitations. Still, LBBD solves to near-optimality all datasets, for which feasible solutions are provided. That gap is less than 10% for up to 10 machines and no more than 20%. There is an obvious trade-off between Gap and Time, as very few datasets are solved in less than one hour. For the first set of α values, we observe that the difference between the gaps achieved by the LBBD and GHA is insignificant. For datasets of 2 machines, LBBD computes worse solutions than GHA in average. As α values increase, i.e., setup and process times costs are increased, LBBD tends to provide significantly improved solutions. Overall, as expected for an exact solution method, LBBD provides improved solutions with an average gap of 5.44% compared to the 17.57% of GHA. The average duration of LBBD, though, is larger than one hour, while the heuristic solution is provided in a few minutes. Combined together, they form a versatile approach for finding solutions of good quality in instances of substantial size.

Instance		Machines															Mean		
		2			5			10			15			20					
		Gap	Time	GHA	Gap	Time	GHA	Gap	Time	GHA	Gap	Time	GHA	Gap	Time	GHA	Gap	Time	GHA
J	10	0.89	< 1	4.64	1.23	5	10.65	3.82	1488	11.00	6.30	3507	17.02	10.86	6026	14.99	4.62	2205	11.66
	20	0.51	< 1	2.41	1.98	11	6.57	3.96	5755	18.25	9.87	6940	20.59	10.94	7338	31.48	5.45	4009	15.86
	30	0.39	1	2.20	1.73	83	8.71	5.16	4430	26.25	10.29	7035	30.20	10.63	7781	39.39	5.64	3866	21.35
	40	0.57	< 1	1.04	2.13	260	6.31	5.72	5113	20.62	11.47	6809	32.57	15.42	7065	47.20	7.06	3850	21.55
	50	0.49	9	0.98	1.48	3479	5.84	3.81	8740	17.86	9.47	7537	36.30	19.69	7274	54.54	6.99	5408	22.90
	80	0.29	6	1.26	1.39	232	3.06	2.87	7545	18.06	9.35	8019	35.50	18.98	8221	53.82	6.58	4804	22.34
	100	0.39	11	0.91	1.52	196	3.10	2.64	6879	14.64	7.03	6578	34.33	17.57	7952	48.99	5.83	4323	20.39
	150	0.40	41	0.83	1.27	334	3.09	2.58	7836	13.99	5.93	8541	27.90	13.67	10109	46.17	4.77	5372	18.20
	200	0.46	392	0.58	1.26	2645	2.85	2.33	8679	11.58	5.47	8925	28.64	17.95	7955	44.28	5.49	5719	17.59
	300	0.44	597	0.56	1.18	5312	2.79	7.40	8185	14.97	-	-	-	-	-	-	3.01	4698	6.10
	400	0.43	559	0.66	-	-	-	-	-	-	-	-	-	-	-	-	0.43	559	0.66
	500	0.44	689	0.54	-	-	-	-	-	-	-	-	-	-	-	-	0.44	689	0.54
	700	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	1000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
α	[0.01, 0.1]	0.88	170	0.33	2.53	3029	2.85	4.19	7944	4.34	6.15	7424	7.16	4.92	5891	6.99	3.56	4758	3.55
	[0.1, 0.2]	0.23	193	1.19	1.59	1279	6.58	4.03	6299	17.98	6.46	6949	28.82	12.32	7068	41.69	4.21	4117	17.68
	[0.1, 0.5]	0.32	214	2.63	1.62	642	11.12	5.90	4532	28.64	14.30	6885	51.26	20.06	6239	57.39	8.55	3913	31.49
R	1	0.58	196	1.81	3.60	2165	20.60	11.99	7324	49.59	29.93	8157	89.69	25.46	5827	67.64	10.97	4740	37.55
	3	0.42	185	1.17	1.32	1142	4.70	3.14	6168	10.09	4.44	7523	20.79	4.35	5945	18.48	2.70	4165	8.79
	5	0.42	196	1.17	1.32	1141	4.35	2.94	5680	8.23	4.27	6531	12.49	5.67	7598	13.70	2.65	3883	6.38
Mean		0.48	192	1.38	2.46	2257	11.24	6.78	6667	24.51	14.73	7260	44.23	15.08	7747	42.21	5.44	4262	17.57

Table 9: Benchmark Results of LBBD for Weaving Scheduling - Makespan

4.4.2. Benchmark experiments for Weaving Scheduling – Tardiness

The benchmark experiments for Weaving Scheduling – Tardiness were performed on a Linux server (4 processors, 3.3 GHz CPU, 12 GB RAM) using CPLEX 20.1 (Python API for MILP and CP Optimizer in DOcplex for CP). A time limit of 300 seconds is imposed on the master problem M. For the termination of Algorithm 3, a maximum number of K = 20 iterations and a maximum Gap of E = 0.1% are imposed. We consider randomly generated instances of $|J| \in \{10, 20, 50, 100, 150, 200\}$ jobs and $|M| \in \{2, 5, 10, 20\}$ machines. To the best of our knowledge, the maximum-sized among these exceed the sizes that are presented in relative literature for exact methods. Each combination of $|J| \times |M|$ (excluding instances for which $|M| \geq |J|$) is run for $R = \{2, 5, 10, 20\}$. If $R = |M|$, there are unlimited resources in practice, instances for which $R > |M|$ are meaningless. We also consider two variations of tardiness, denoted by Tight and Loose attributes. Processing and Setup times are created as for benchmark instances for Weaving Scheduling – Makespan, while $\tau = \{0.2,0.8\}$ and $\rho = \{0.2,0.8\}$. $P_i = \min_{j,m} (p_{jm} + s_{ijm})$ and d_i is selected u.a.r from $(P_i \cdot (1-\tau - \frac{\rho}{2})$, $P_i \cdot (1-\tau + \frac{\rho}{2}))$. Note that for Tight instances, $\tau = 0.2$, $\rho = 0.8$, and for Loose instances, $\tau = 0.8$, $\rho = 0.2$. The generators for the processing times, setup times, and due times are inspired by Fotakis et al. (2016), Kim & Lee (2021) and Dogramaci & Surkis (1979) respectively. The results are presented in Table 10.

$\|J\|$	$\|M\| = 2$		$\|M\| = 5$		$\|M\| = 10$		$\|M\| = 20$	
	Gap(%)	Time(s)	Gap(%)	Time(s)	Gap(%)	Time(s)	Gap(%)	Time(s)
10-20	3.24	3	2.11	4	6.68	22	-	-
50-100	3.84	34	4.11	106	6.29	1025	14.05	2051
150-200	3.82	100	3.95	410	5.35	1106	15.38	2494
D	$\|M\| = 2$		$\|M\| = 5$		$\|M\| = 10$		$\|M\| = 20$	
	Gap(%)	Time(s)	Gap(%)	Time(s)	Gap(%)	Time(s)	Gap(%)	Time(s)
L	3.42	47	3.16	177	5.35	878	13.37	2385
T	3.84	45	3.62	170	6.63	835	16.06	2160
R	$\|M\| = 2$		$\|M\| = 5$		$\|M\| = 10$		$\|M\| = 20$	
	Gap(%)	Time(s)	Gap(%)	Time(s)	Gap(%)	Time(s)	Gap(%)	Time(s)
2	3.63	46	3.80	221	9.26	1773	41.24	2831
5	-	-	2.98	126	4.75	468	8.09	2825
10	-	-	-	-	3.96	328	5.40	2410
20	-	-	-	-	-	-	4.14	1023

$D : L$ for *Loose* and T for *Tight* deadlines, LB : Best Lower Bound, UB : Best Upper Bound, Time(s) : Duration of the algorithm in seconds, Gap(%) : $100 \cdot \frac{UB - LB}{UB}$

Table 10: Results for Weaving Scheduling – Tardiness on benchmark instances

As we notice, the value of gap is less than 10% for most of the generated instances. Tight deadlines imply solutions of smaller gaps than the respective Loose ones, nevertheless both cases are solved in almost identical times. However, we notice that the limitation of resources has a heavy impact on the performance of the LBBD, as the gaps and the elapsed time are both significantly increased. For instances of R = 2 workers and |M| > 10, the algorithm requires almost one hour to be completed and the computed gaps are greater than 10%. For the rest of the instances, no more than a few minutes are required to terminate the algorithm.

4.5. Results on PIACENZA data

We, now, present our results for experiments performed on real data from PIACENZA.

4.5.1. Results of Weaving Scheduling – Makespan

We have applied our methods on the weaving plant of PIACENZA. The plant operates with two different types of (parallel) looms depending on their corresponding speed, namely 300 or 400 strokes/minute. This information, combined with fabric specification and the quantity of an order, returns the processing time needed per job and loom. Since the aforementioned calculated time is an estimation, we include in our experiments a factor c_{im} u.a.r. selected from [0.8, 1.2] to better capture the real nature of the problem. The values of sequence-dependent setup times are taken from {0h, 4h, 6h, 8h}, and are determined by the chainability/annotability codes, yarns and comb heights, and codes for each job (s_0 = 1h). There are 3 groups of workers available for setups, each loom requiring one such group, plus an upper bound of 50h on the daily total setup time. Each order is splittable to parts of length at least 50 meters. Let us list some minor adjustments of our approach. To apply GHA, we substitute constraints (10) in both (MIP 1) and (MIP 2) with the following:

$$\ell_{im} \cdot y_{im} \leq w_{im} \cdot p_{im} \qquad \forall i \in J, m \in M. \qquad (49)$$

In (MIP 1), we also need to include:

$$\ell_{im} \cdot z_{im} \leq f_{im} \cdot p_{im} \qquad \forall i \in J, m \in M, \qquad (50)$$

where the parameter ℓ_{im} is calculated by the processing time calculation formula, when setting the quantity equal to the 'lower splitting bound' (50m in our case). To adapt our LBBD accordingly, we add to the master problem M^{k-1} of iteration k − 1:

$$\ell_i \cdot y_{i,m}^{k-1} \leq \frac{p_{i,m}}{100} \cdot W_{i,m}^{k-1} \qquad \forall i \in J^*, m \in M. \qquad (51)$$

To enforce the maximum of 50h on daily setup time, we add to the subproblem S_k of iteration k:

$$\sum_{m \in M_{k-1}} \sum_{i=\{1,|\nu_m^{k-1}|\}} (\texttt{length\_of}(\mu_{i,m}^k) | d - 1 \leq \texttt{end\_of}(\mu_{i,m}^k) \leq d) \leq u \quad \forall d \in D, \qquad (52)$$

where u = 3000 (50h in minutes) and D is the cumulative minutes per day, e.g., D = {1440, 2880, 4320} regarding 3 subsequent days. The experiments are performed on 45 weekly instances, spanning January to December 2020. The number of jobs per instance ranges from 7 to 94. GHA and LBBD results are presented in Table 11.

# Orders	% Gaps of GHA						Mean Gap (%)	Mean t(s)	% Gaps of LBBD						Mean Gap (%)	Mean t(s)
	2	4	6	8	10	12			2	4	6	8	10	12		
[7,32]	11.59	14.29	17.32	17.48	22.91	23.71	17.88	19	0.19	0.32	0.92	1.55	2.23	3.53	1.46	1539
[35,44]	8.51	10.03	13.45	15	19.66	21.75	14.73	18	0.14	0.40	0.65	1.13	2.00	3.67	1.33	4590
[45,51]	11.94	17.61	19.83	23.76	30.43	44.96	24.76	13	0.32	0.72	1.02	1.70	6.16	6.49	2.74	5501
[56,61]	10.92	14.9	17.87	23.27	26.71	35.06	21.46	8	0.23	0.56	0.92	1.77	6.81	3.57	2.31	5098
[65,94]	11.9	16.56	19.66	24.42	31.05	46.9	25.08	33	0.14	0.55	0.91	1.73	8.30	6.66	3.05	5873
Mean Gap	11	14.74	17.67	20.79	26.23	34.7	20.86	-	0.21	0.51	0.88	1.58	5.09	4.85	2.19	-
Mean t(s)	< 1	5	12	29	22	41	-	18	448	765	2749	5901	8680	8630	-	4529

Table 11: Results for Weaving Scheduling – Makespan on real data

GHA reaches a good solution in a few seconds, while LBBD establishes near-optimality in less than 2 hours.

4.5.2. Results of Weaving Scheduling – Tardiness

We now present the results of the algorithm presented for Weaving Scheduling – Tardiness on real data instances.

# Orders	% Gaps for $\sum_j T_j$		Mean Gap (%)	Mean t (s)	% Gaps for $\sum_j U_j$		Mean Gap (%)	Mean t (s)
	20	30			20	30		
[7, 32]	3.73	8.09	3.71	593	2.78	2.78	2.78	167
[35, 44]	3.25	5.92	3.35	841	0.00	0.00	0.00	6
[45, 51]	12.11	2.43	12.89	1407	22.85	22.85	22.85	810
[56, 61]	8.28	4.10	8.25	1081	17.05	10.00	13.52	232
[65, 94]	13.58	22.04	13.16	1315	23.60	21.16	22.38	566
Mean Gap (%)	8.19	8.52	8.27	-	13.25	11.36	12.30	-
Mean t(s)	1056.0	1053	-	1047	373	374	-	356

Table 12: Results for Weaving Scheduling – Tardiness on real data

We observe that the time needed has severely dropped for both objectives (total tardiness and number of tardy jobs) compared to the makespan objective. This mainly comes from the fact that job splitting – a property that highly increases computational complexity – is ignored. However, we observe that our method would be able to be applied on an online manner for the PIACENZA environment.

4.6. Decision Support

Apart from efficiently solving incoming instances, it is important to provide the necessary decision support for plant managers.

4.6.1. Sensitivity Analysis

Given this performance, we have used GHA as a tool evaluating different structural decisions, namely the splitting lower bound SLB, the daily setup limit DSL and the number of worker groups R available for setup, i.e., the maximum number of simultaneous setups. Please note that LBBD could be used for the same purpose but under a significantly longer computational time. We divide the weekly instances into two sets using either the number of jobs |J| or the average order quantity |Q| in meters or the percentage of splittable jobs h. Let N = {1, 2,..46} denote the set of real data instances. Averages over all 46 weekly instances are as follows: the mean number of jobs is |J| = 49.5, the mean order quantity is k = 123.39 m, calculated by $k = \sum_{n=1}^{N} \frac{|Qn|}{N}$ where $|Q|_n$ is the average order quantity of instance $n \in N$, and the average percentage of splittable jobs is h = 34.7 % calculated by $h = = \sum_{n=1}^{N} \frac{Split\%n}{N}$ where $Split\%_n = \frac{\text{Splittable\_jobs}}{|J|n}$ is that percentage regarding instance n. For each division, we have solved both subsets of the 46 instances for all 27 combinations of R = {1, 3, 5}, DSL = {3000, 4500, 6000} and SLB = {30, 50, 70}, over M = {2, 4, 6, 8, 10, 12}. For each combination and number of machines we calculate the average makespan. As changes in SLB show a small impact, we sustain only the makespan for SLB= 50m. Table 13 shows the percentage of makespan change when moving from R = 1 to R = 3, or R = 3 to R = 5 on each criterion group. IR as a prefix denotes any (average) makespan change attributed to the number of worker groups R. We observe that as the number of machines

increases extra workers are needed. It is also evident that an increase in the number of working groups R from 1 to 3, has a substantial impact, which is diminished when moving from R = 3 to R = 5. This is more impressive for large instances, smaller-sized orders, and low splittability. These are valuable insights for the plant managers regarding the investment in human resources depending on the characteristics of the weekly orders and the number of looms available.

Crit.	R	Machines						Mean
		2	4	6	8	10	12	
$J \leq 49$	$IR_{1->3}$	-1.33	-5.97	-13.26	-19.57	-26.02	-31.48	-11.09
	$IR_{3->5}$	0	-0.01	-0.34	-0.72	-1.25	-2.61	-0.41
$J > 49$	$IR_{1->3}$	-2.6	-11.41	-23.05	-32.91	-40.25	-43.92	-19.42
	$IR_{3->5}$	0	-0.04	-0.28	-1.58	-2.38	-3.97	-0.7
$k \leq 123.39$	$IR_{1->3}$	-3.32	-13.64	-27.73	-37.22	-44.53	-46.69	-22.68
	$IR_{3->5}$	0	-0.01	-0.44	-1.88	-3.04	-5.19	-0.91
$k > 123.39$	$IR_{1->3}$	-1.06	-5.28	-10.72	-17.87	-23.93	-31.09	-9.94
	$IR_{3->5}$	0	-0.05	-0.2	-0.68	-0.95	-1.83	-0.31
$h \leq 34.7\%$	$IR_{1->3}$	-2.66	-12.57	-26	-36.27	-42.49	-45.26	-21.31
	$IR_{3->5}$	0	-0.02	-0.43	-1.8	-2.85	-4.72	-0.85
$h > 34.7\%$	$IR_{1->3}$	-1.55	-5.99	-11.99	-18.16	-26.09	-32.43	-10.87
	$IR_{3->5}$	0	-0.05	-0.2	-0.69	-1.03	-2.16	-0.34

Table 13: Improvements of R vs M under different criteria

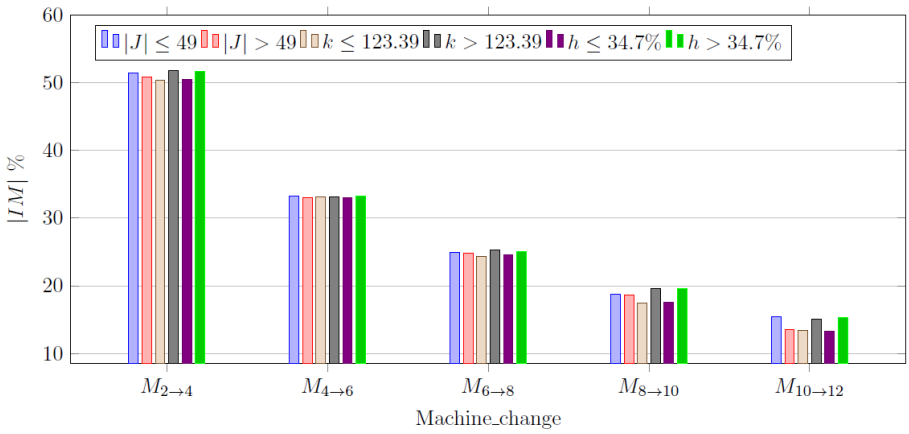

Figure 8: Solution changes when increasing |M| under different criteria

Analogous insights are provided regarding the number of machines in Figure 8. IM is the makespan change observed when the 'control' variable is the number of machines. Indicatively, moving from |M| = 2 to |M| = 4 slashes the makespan by 50 %. This improvement is, as expected, smaller as M increases further. Although the change is approximately identical regardless of the three criteria dividing the instances (jobs, order size, splittability), larger quantities and more splittable orders exhibit a slightly larger improvement, as this further exploits the availability of additional machines. Here, a plant manager may decide upon the trade-off between higher capital investment versus higher operating cost. Last, the impact on makespan when increasing DSL (the daily setup limit) is shown in Figure 9 and Figure 10. For R = 3, |M| = 12 and increasing DSL from 3000 minutes to 4500 minutes, we have important improvements of 5% especially for large size instances with small quantities and mostly unsplittable jobs. However, these improvements do not significantly change when DSL is set to 6000 minutes. To the contrary Figure 10, shows that for R = 5 and |M| = 12, significant improvements remain when DSL increases from 4500 to 6000. A cost-benefit analysis for the different investment options is beyond the scope of our study. What the above show is that an exact method that consistently finds near-optimal solutions in reasonable time can be nicely exploited as a decision-support tool for both operational decisions (e.g., increase of resources) or more strategic ones (e.g., increased production capacity).

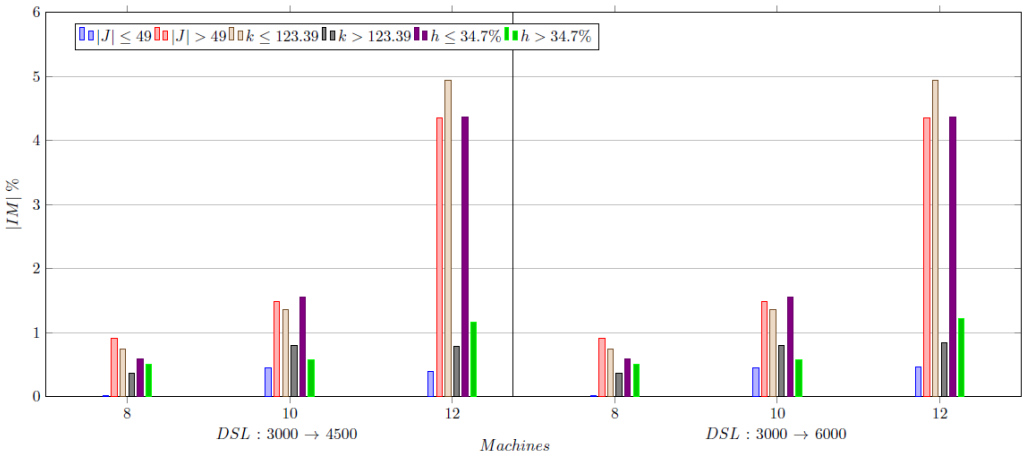

Figure 9: Solution Improvements when increasing DSL for R = 3

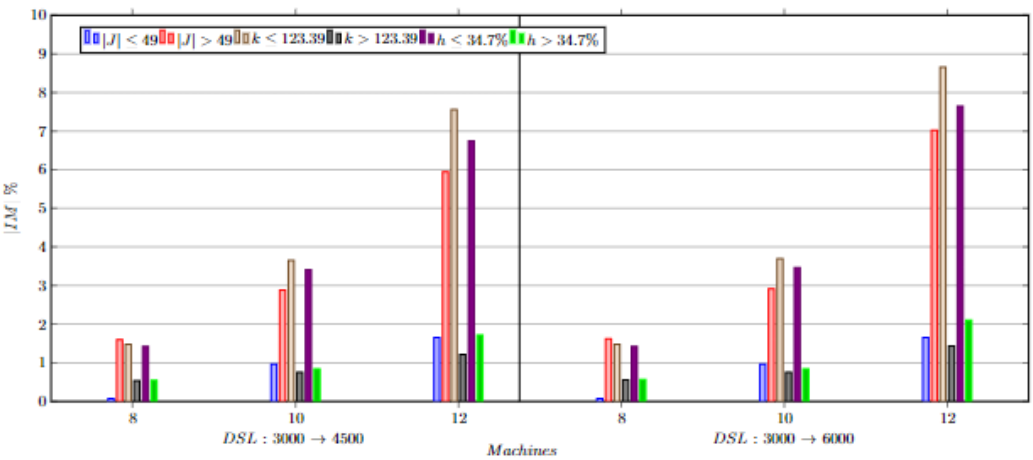

Figure 10: Solution Improvements when increasing DSL for R = 3

On a modern business environment, it is often the case that unexpected events occur that negatively impact the production process. In order to present our findings, we use the following example, with an instance created in a similar manner as in Section 4.4.1. Suppose we are working on an environment, which operates with three unrelated machines and a set of seven customer orders (jobs) is received. Processing times and setup times are provided on the following tables (measured in time units). Setup resources are assumed to be unlimited.

J \ J	1	2	3	4	5	6	7
1	0	1	1	1	1	0	2
2	1	0	2	2	2	1	1
3	3	2	0	3	1	1	1
4	2	2	1	0	2	1	2
5	2	1	2	1	0	1	1
6	1	1	1	1	1	0	1
7	2	2	3	3	1	3	0

J \ J	1	2	3	4	5	6	7
1	0	1	1	1	1	0	1
2	1	0	1	1	1	1	1
3	1	2	0	2	2	2	2
4	1	3	2	0	3	1	4
5	1	2	1	1	0	1	2
6	1	1	2	2	1	0	3
7	1	1	1	2	1	1	0

J \ J	1	2	3	4	5	6	7
1	0	0	1	2	1	1	1
2	1	0	1	1	1	1	1
3	1	1	0	1	2	1	2
4	1	1	1	0	1	3	3
5	1	2	2	2	0	1	1
6	1	1	1	1	1	0	1
7	1	2	2	2	3	2	0

J \ M	1	2	3
1	4	3	4
2	4	3	2
3	7	5	6
4	6	8	6
5	6	4	4
6	5	6	4
7	6	5	6

Table 14: Example Instance

The optimal solution is presented on the Figure 11.

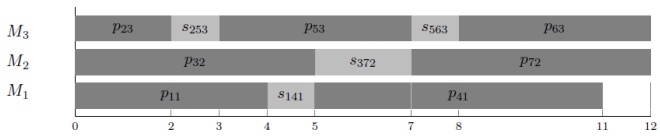

Figure 11: Gantt chart for optimal solution for Example Instance

4.6.2. Maintenance

One case that may occur is when machines need to be shut down e.g. for maintenance purposes. When this information is available by the time a production schedule has to be provided, the optimization algorithm has to take into consideration time intervals for which, a machine will be unavailable. Thus, we would recommend the inclusion of two additional input fields for each loom, providing information regarding the time that a machine may be unavailable. Now, let us assume that beforehand, it is known that machine 1 will be unavailable on time units 7 to 10. A near-optimal solution in this case is presented in Figure 12.

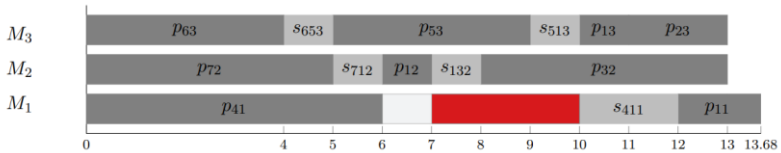

Figure 12: Gantt chart for solution of instance with maintenance interval

When comparing the above Gantt chart with the initial one, we observe that the makespan is increased by 1.8. However, the schedule presented above respects the extra constraint of unavailability for machine 1. Thus, it is important for exact methods to be employed, as among others, have the advantage of incorporating naturally these types of restrictions.

4.6.3. Malfunction

On the other hand, a common phenomenon are machine malfunctions. These events occur after the beginning of the execution of the production schedule. Thus, when malfunctions happen, it is necessary to take fast and efficient decisions to normalize the production schedule. We, now, assume that we have begun executing the schedule presented on the initial instance, when suddenly, machine 1 stops the processing and is incapable to recover. The solution of this occasion is shown in Figure 13.

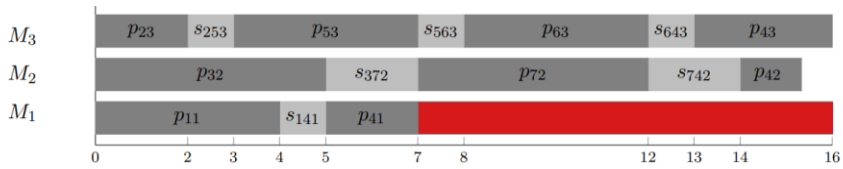

Figure 13: Gantt chart for solution of instance with machine malfunction

We observe that job 4 is forced to stop being processed on machine 1, and is allocated to machines 2 and 3 respectively.

4.7. Concluding remarks

We have examined a parallel machine scheduling problem with unrelated machines, job splitting, sequence- and machine-dependent setup times and limited setup resources. We have proposed a LBBD formulation, which when enhanced with the solutions of a novel heuristic algorithm solves near optimally large-size instances under reasonable time. Our heuristic alone provides good quality solutions for very large instances; solution quality is verified using also the novel lower bounds we propose. Additionally, we have proposed an efficient LBBD approach in order to deal with our problem with the goal of tardiness optimization. The performance of our methods has been evaluated on both various benchmark instances and real data from PIACENZA's weaving process. Our results could motivate further work in various terms. There might be slight variants of the problem examined here, for which both GHA and LBBD might improve current literature. An indicative example is the problem where some orders have different processing alternatives (Kim et al. 2020), thus splitting comes up of the multiple parts of an order, known in advance. Perhaps the most important direction would be to further examine robustness, either through extended formulations that proactively encompass uncertainty for some of the parameters (e.g., the maximum number of simultaneous setups, or machine availability), or through other methods that reactively alleviate the effects of disruptive events (e.g., machine malfunctions or breakdowns), as in (Eirinakis et al. 2021; Kim et al. 2020).

5. Steel Manufacturing: Pilot Case by BRC

5.1 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.

5.1.1 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 practice.

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.

5.1.2 Case Description for BRC

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 14) 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.

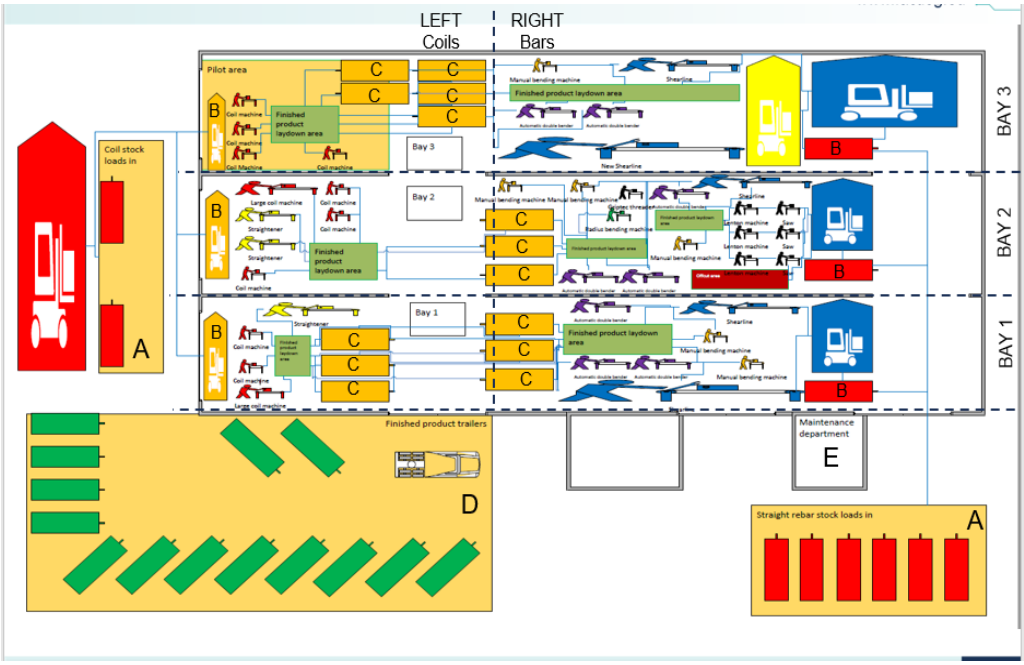

Figure 14: BRC Facility Layout

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 of 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 Total Lateness L_i of the jobs that are tardy. Those KPIs are encoded as follows:

- **Makespan (C_{max})**: one of the most common objective criteria. 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).
- **Maximum Lateness (L_{max})**: The Maximum (Total) Lateness is a measure that is quite often whether the company has very tight due dates in compare with the release and processing times. The maximum lateness of a job i in a schedule is the difference between its completion time C_i minus its due date d_i ($L_i = C_i - d_i$) and is defined as L_{max} = (L_1,....,L_n). In fact, if a job is completed before its due date, its lateness can be negative. So this criterion measures the worst violation of the due dates. 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.

5.2 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 lateness tolerance (so tardy jobs are allowed) and the scale of the problem itself. Sometimes the company has some tight orders' due date so it's inevitable to avoid the job lateness. In the above case our objective is to minimize as much as possible the lateness 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 (on the previous deliverable 5.1) however we can be compatible with this requirement on this 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.

5.3 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. The difference in the current model in compare with the previous one lies on the fact that the cranes' times are incorporated. Due to shortage of real data this modification is on a beta version but it captures with a more realistic way the BRC's production line. At this phase the cranes' movement are treated as parameters which influence the job assignments on the different machines. The cranes re positioning are beyond the scope of the current model as it assumed that every time a crane is needed it's available. An extension of the model can be explored in more depth in future research.

5.3.1 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. The notation has evolved so as to capture the crane times as well.

Sets

- S : set of stages ($s = |S|$)
- $M(s)$: set of machines in stage $s \in S$
- I : set of jobs
- $\hat{B}$: set of forbidden (job,machine) combinations
- $\hat{M}$: set of non-existing processing paths
- $L(s)$: set of valid starting points for loading in each stage $s \in S$
- $M(s)$: set of valid ending points for unloading in each stage $s \in S$

Parameters

- T_i^d : due date of job $i \in I$
- T_i^r : release date of job $i \in I$
- T_m^s : setup time for machine $m \in M(s), s \in S$
- T_{mi}^p : processing time in machine $m \in M(s)$ for job $i \in I$
- U : a positive sufficient large number, $U = \sum_m \sum_i T_{mi}^p$
- λ : weight coefficient between makespan-lateness
- $Z_i = \begin{cases} 0, & \text{iff job i is one stage process} \\ 1, & \text{iff job i is two stage process} \end{cases}$
- l_{lm} : loading time from location $l \in L(s)$ to machine $m \in M(s)$
- u_{mk} : unloading time from machine $m \in M(s)$ to location $k \in M(s)$

Decision Variables

- y_{mi} : 1 iff on machine $m \in M(s)$ we assign 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 $i \in I$, $L_i = max\{c_{is} - T_i^d\ ,\ 0\}$
- T_i : 1 iff job $i \in I$ is tardy and 0 otherwise

5.3.2 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
- At any time a crane is needed is available and no re-positioning time is taking into consideration

5.3.3 Mathematical Formulation

In this section we present our Mixed Integer Quadratic Programming (MIQP) 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.

After discussions with BRC we established as criterion a convex sum of makespan and the total lateness of tardy jobs, see (5.1). As we can observe to incorporate the crane movements we need to "jump" from the Mixed Integer Linear Programming to a Mixed Integer Quadratic Programming. The main component of this model is the fact that the crane movements can affect the job assignments among the machines. The challenge is that every time the model should be able to know the starting and the ending point of the crane. The decision for a job assignment on a machine requires the investigation of all the potential movements for feeding and unloading this machine. The idea itself hides something that is not linear however we can linearize it using different methodologies.

We define Y_{mi} = 1 if job i is assigned on machine m. The set of constraints (5.2) 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 (5.3) link the makespan decision variable with the completion time of the last job to finish its processing. The next four sets of constraints (5.4) - (5.7) are related with the completion time of a job. More precisely constraint (5.4) 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 (5.5) 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.

$$minimize \quad \lambda C_{max} + (1-\lambda)\sum_{i\in I} L_i \tag{5.1}$$

$$s.t.$$

$$\sum_{m\in M(s)} y_{mi} = 1 \qquad \forall i \in I, s \in S \tag{5.2}$$

$$C_{max} \geq c_{is} \qquad \forall i \in I, s \in S \tag{5.3}$$

$$c_{is} \geq \sum_{m\in M(s)} y_{mi}[T_i^r + l_{lm} + T_m^s + T_{mi}^p + (1-Z_i)\cdot u_{mk} + Z_i \cdot \sum_{m'\in M(s+1)} (y_{m'i}\cdot u_{mk})] \qquad \forall i \in I, s \in S \tag{5.4}$$

$$c_{is} \leq c_{i,s+1} - \sum_{m\in M(s+1)} y_{mi}[T_{mi}^p + T_m^s + (1-Z_i)\cdot u_{mk} + Z_i \cdot \sum_{m'\in M(s+2)} (y_{m'i}\cdot u_{mk})] \qquad \forall i \in I, s < |S| \tag{5.5}$$

$$c_{i's} \geq c_{is} + T_{mi'}^p + T_m^s + l_{lm} + (1-Z_{i'})\cdot u_{mk} + Z_{i'} \cdot \sum_{m'\in M(s+1)} (y_{m'i'}\cdot u_{mk}) - U(3 - y_{mi} - y_{mi'} - x_{ii's})$$
$$\forall i, i' \in I, i < i', s \in S, m \in M(s) \tag{5.6}$$

$$c_{is} \geq c_{i's} + T_{mi}^p + T_m^s + l_{lm} + (1-Z_i)\cdot u_{mk} + Z_i \cdot \sum_{m'\in M(s+1)} (y_{m'i}\cdot u_{mk}) - U(2 - y_{mi} - y_{mi'} - x_{ii's})$$
$$\forall i, i' \in I, i < i', \forall s \in S, m \in M(s) \tag{5.7}$$

$$y_{mi} = 0 \qquad \forall i \in I, m \in M, (i,m) \in \hat{B} \tag{5.8}$$

$$y_{mi} + y_{m'i} \leq 1 \qquad \forall i \in I, (m,m') \in \hat{M} \tag{5.9}$$

$$L_i = \max\{c_{is} - T_i^d, 0\} \qquad \forall i \in I, s \in S \tag{5.10}$$

$$L_i \leq T_i U \qquad \forall i \in I \tag{5.11}$$

$$T_i \leq L_i U \qquad \forall i \in I \tag{5.12}$$

$$\sum_{i\in I} y_{mi}(T_{mi}^p + T_m^s) \leq \max_{i\in I}\{T_i^d - \sum_{s'\in S, s'>s} \min_{m'\in M(s'),\ (i,m')\notin \hat{B}} (T_{m'i}^p + T_{m'}^s)\} - \min_{i\in I}\{T_i^r + \sum_{s'\in S, s'<s} \min_{m'\in M(s'),(i,m')\notin \hat{B}} (T_{m'i}^p + T_{m'}^s)\}, \qquad \forall s \in S, m \in M(s) \tag{5.13}$$

$$x_{ii's}, y_{mi}, T_i \in \{0,1\},\ 0 \leq \lambda \leq 1, c_{is} \geq 0 \tag{5.14}$$

The next constraint (5.5) 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 unloading time is calculated in the same manner as in the previous constraint set.

The next two set of constraints (5.6) - (5.7) prevent any two jobs from overlapping in a common machine. The difference of completion times between job i, which precedes job $\acute{\imath}$ should be at least the setup time plus the processing time of the first plus the loading and unloading times. 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 $xii's$ decision variables because the nature of the constraints and the relationship that exist between ymi 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:

$$(5.6): c_{i's} - T^{p}_{mi'} - T^{s}_{m} - l_{lm} - (1 - Z_{i'}) \cdot u_{mk} - Z_{i'} \cdot \sum_{m' \in M(s+1)} (y_{m'i'} \geq c_{is}, \text{ so starting time } i' \geq \text{finishing time } i \Rightarrow \text{True}$$

$$(5.7)\quad c_{is} - T^{p}_{mi} - T^{s}_{m} - l_{lm} - (1 - Z_{i}) \cdot u_{mk} - Z_{i} \cdot \sum_{m' \in M(s+1)} (y_{m'i} \cdot u_{mk}) \geq c_{i's} - U \Rightarrow \text{trivial}$$

We can check that the first constraint implies that the starting time of job $\acute{\imath}$ ($c_{\acute{\imath}s} - T^{p}_{m\acute{\imath}} - T^{s}_{m} - loading\ time - unloading\ time$) is at least the completion time of job i. However, the second constraint is redundant. So, the initial assumption which job i precedes $\acute{\imath}$ is hold.

Forbidden assignments are specified in (5.8), where $\widehat{B}$ 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 (5.9) 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 (5.10) - (5.11) referred to the lateness and tardiness of a job. In more detail, constraints (5.10) calculate the lateness of a job and specified only the positive lateness as tardiness (Li=max $\{c_{is} - T_{id}, 0\}$. Constraints (5.11 - 5.12) links the lateness and tardiness, namely if and only 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 (5.13) 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 (5.14) ensure the integrallity constraints and the non-negativity as well.

5.4 Computational Results

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 GUROBI solver. We want to emphasize on the statistics as the scale of the instances raise. Due to shortage of real data, we created batches of test instances which 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)	lateness (sec)	makespan-lateness (sec)
(n=10, m=9)	0.85 (100%)	2.16 (100%)	1.80 (100%)
(n=12, m=9)	1.44 (100%)	13.76 (100%)	56.33 (100%)
(n=14, m=9)	77.7 (90%)	5.64 (60%)	72.43 (70%)
(n=16, m=9)	259.70 (90%)	47.85 (50%)	447.90(70%)
(n=18, m=9)	113.75 (50%)	(0%)	128.93(50%)
(n=20, m=9)	89.45 (50%)	(0%)	- (0%)

Table 15: Instances' solution times 1

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-lateness jobs is the most computational expensive case having big solution times and small percentage of solvability.

6. Automotive Manufacturing: Pilot Case by CONTINENTAL

6.1 Introduction

Continental is major manufacturer for automotive parts that are used in the assembly of dashboard shells. Key decision areas and pain points 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 part of the production line examined in the context of FACTLOG consists of two stages. The first stage is preassembly subline that consists of 5 process steps, followed by one buffer step. Afterwards, the second stage contains 18 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). Thus, 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 production setting can be modelled as a multi-stage flow shop scheduling problem (FSSP) with resource constraints (e.g. semi-finished products, raw materials etc). 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 that minimizes the completion time of the latest job (makespan), followed by the total job tardiness (delay from the deadline date).

Apart from resource and other operational constraints, another important dimension is the scheduling of maintenance activities. It is 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.

In this project, a Constrained Programming (CP) approach has been adopted for modelling and solving this integrated 2-stage Flow Shop Scheduling Problem with Resource Constraints and Maintenance Windows. The optimization model takes as 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.

The core functionality provided to the human planners in this project is the ability to generate static plans and perform what-if scenarios for maintenance activities. For example, the planner can generate plans for different sets of production orders, different resource availability and different processing times and production volumes. The human planner can generate plan with different maintenance windows or impose specific maintenance tasks.

As described earlier in this report, the production scheduling engine can be accessed via a Restful API that accepts requests and delivers responses using Hypertext Transfer Protocol (HTTP) and JSON text format for data exchange. The input and output protocol and the format of the JSON files are described in Deliverable 5.1. The production schedules produced in the output can be also depicted for further evaluation by the associated simulation tools.

Although it has not been tested and validated in this round of computational experiments, we have also developed during this project the ability to update a given 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 and provides the state of the master baseline plan (i.e. production orders completed and in progress), the optimization engine produces a new re-optimized plan. This functionality has not been tested with real production data given the difficulty to maintain manually the current state of the production lines by the human planners.

Below, Section 6.2 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 6.3 describes in detail the optimization model. Finally, Section 6.4 provides some computational experiments on synthetic benchmark data sets as well as real production data.

6.2 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.

The table below 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

Reference	Problem type	Maintenance type	Machine degradation	Notes	Objective	Stochastic aspect	Method
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)
Shijin and Jianbo (2010)	Flexible Job-Shop Schedulin g	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

Reference	Problem type	Maintenance type	Machine degradation	Notes	Objective	Stochastic aspect	Method
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)
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

Reference	Problem type	Maintenance type	Machine degradation	Notes	Objective	Stochastic aspect	Method
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 16: Literature review for integrated shop scheduling and maintenance planning

6.3 Model formulation and solution method

6.3.1 Notation

The examined 2-stage FSSP with resource constraints and maintenance activities is a special case of the more generalized FJSSP with resource constraints which is described in the rest of this section. 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, $\Omega = \cup_{u=1}^{l} O_u$. 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_{max}^s, is defined as the maximum completion time of all operations in Ω.

6.3.2 Constraint Programming Formulation

CP 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
- $AlwaysIn(R,a,v_{min},v_{max})$, i.e., the value of the cumulative function R during the time that interval variable a is active should be greater or equal to v_{min} and less or equal to v_{max}

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 $\phi^b_{i,k}$ is defined. The maximum capacity of the limited capacity buffer of a machine i is denoted by lcb_i, while the cumulative function that is used to accumulate the consumption of the buffer overtime is denoted by LCB_i. 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 $\sigma_k = \{\phi_{i,k}, \forall i \in \Omega\}$.

$$\min C_{max} \tag{6.3}$$

subject to:

$$Alternative(\tau_i, \mu_i) \forall i \in \tag{6.4}$$

$$EndBeforeStart(j,i) \forall i \in \Omega, \forall j \in PJ_i \tag{6.5}$$

$$NoOverlap(\sigma_k) \forall k \in M \tag{6.6}$$

$$\sum_{j=1}^{n} Pulse\left(\phi_{j,k}, u_i(k)\right) \forall k \in M_j \leq \bar{U}_i \forall i \in U \tag{6.7}$$

$$\sum_{j=1}^{n} Pulse\left(\tau_j, t_i(j)\right) \leq \overline{T_i} \forall i \in T, t_i(j) = i \quad (6.8)$$

$$Pulse\left(\phi_{j,k}^{b}, 1\right) \forall j \in \Omega, \forall k \in M_j \quad (6.9)$$

$$AlwaysIn\left(LCB_k, \phi_{j,k}, 0, lcb_k\right) \forall j \in \Omega, \forall k \in M_j \quad (6.10)$$

$$PrecenseOf\left(\phi_{j,k}^{b}\right) = PresenceOf\left(\phi_{j,k}\right) \forall j \in \Omega, \forall k \in M_j \quad (6.11)$$

$$StartOf\left(\phi_{j,k}^{b}\right) = EndOf\left(\phi_{j,k}\right) \forall j \in \Omega, \forall k \in M_j \quad (6.12)$$

$$EndOf\left(\phi_{j,k}^{b}\right) = PresenceOf\left(\phi_{j,k}^{b}\right) StartOf\left(\tau_{sj_j}\right) \forall j \in \Omega, \forall k \in M_j \quad (6.13)$$

$$Step\left(0, R_i^{start}\right) + \sum_{j \in \Omega | Prod(j)=i} StepAtEnd\left(\tau_j, lotSize\right) - \sum_{j \in \Omega | Req(j)=i} StepAtStart\left(\tau_j, lotSize\right) \geq 0 \; \forall i \in R \quad (6.14)$$

$$\sum_{i \in R | wip(i)=k} \left\{ Step\left(0, R_i^{start}\right) + \sum_{j \in \Omega | Prod(j)=i} StepAtEnd\left(\tau_j, lotSize\right) - \sum_{j \in \Omega | Req(j)=i} StepAtStart\left(\tau_j, lotSize\right) \right\} \leq \overline{W_k} \forall k \in W \quad (6.15)$$

$$C_{max} \geq EndOf\left(\tau_i\right) \forall i \in \quad (6.16)$$

The objective (6.3) refers to the minimization of the makespan. Constraints (6.4) are used to enforce a unique selection of the available modes for the interval variable τ_i out of the set μ_i. Constraints (6.5) 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 (6.6) 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 (6.7) and (6.8) 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 (6.9, 6.10, 6.11, 6.12 and 6.13) are used to describe the usage of limited capacity buffers. More specifically, constraint (6.9) defines the occupation of the buffer, while constraint (6.10) makes sure that at start of any activity on the machine there are no other activities already waiting in the buffer, so that the buffer capacity is violated. Constraints (6.11), (6.12) and (6.13) are used to calculate the start and end times of the decision interval variables related to the limited capacity buffers. Constraints (6.14) are used to accumulate the production and consumption of each generalized resource, while constraints (6.15) accumulate the usage of resources on their corresponding work in progress buffer. Lastly, constraint (6.16) 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 windows). These dummy jobs have zero processing time on all machines / workstations, except the one that maintenance will be performed. In addition, to emulate a flow-shop environment, $samesequence$ global constrained is used to enforce that the execution order of jobs is maintained for all the workplaces of a particular production line.

As part of the research work performed for this project and to approach very-large scale problem instances, the above CP approach has been also utilized within a more advanced

population-based algorithmic framework. The proposed hybrid CP algorithm uses frequency-based learning mechanisms to detect promising regions in the solution space. The extracted information is used to guide the CP towards finding high quality solutions in short computational times. We benchmark our algorithm on well-known instances of the FJSSP literature with and without resource constraints. Computational results show the effectiveness of the proposed algorithm compared to the basic CP non-hybrid approach. These results will be presented at the MIM Conference together with an extensive analysis on the effect of different types of resource constraints.

6.4 Computational Experiments

6.4.1 Experiments on synthetic benchmark data sets

Initially, we assess the impact of resources constraints on the production schedules. 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 (2007) 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 17. In the second step, 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. The results of the second step are presented in Column (RC_1) of Table 17.

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

The last Column of Table 15 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 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 18 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 can 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 18: Results on small and large scale Flexible Job Shop Scheduling Problems

6.4.2 Experiments on real data

As described earlier, Continental production floor consists of a pre-defined/fixed sequence of process stages that is followed by all products across multiple assembly lines. Assembly lines consist of linear sequences of machines/stations that products progress through. There are various operational realties that needs to be considered:

- Multiple product families with different setup times (setup times based on product family changes on the assembly line)
- Resource Constraints
  - Components (raw materials) required for production or assembly operations
  - Bill of materials, i.e. several semi-finished products are required for the assembly of a product.
  - Work-in-progress areas between production stages with limited capacity
- Fixed/Predicted Maintenance activities
  - Fixed maintenance activities are used to describe already known (pre-scheduled) maintenance activities that will take place as well as predicted maintenance activities for which a time window is only given. The goal is to schedule maintenance activities to the most convenient times for the overall schedule.
- Due dates for production orders

We model the above assembly scheduling environment as a Multi-stage Flow-Shop Scheduling Problem with Resource Constraints and Sequence Dependent Setup Times. Let a set of jobs to complete (J_k), a set of sub-operations for each Job (O_{ij}) and a set of machines where operations are executed (M_i). Each operation can be executed on a single machine with a known processing time (p_{ij}). The number of operations for all jobs is the same and equal to (|M|). Finally, let operation O_{ik} denote the execution of operation i on machine M_k. The goal is to schedule maintenance activities (scheduled or predicted) and production orders such that the makespan or objectives is minimized.

Like the experiments conducted earlier using synthetic data, we initially try to measure the effect of maintenance activities and resources constraints using real sample data provided by Continental. Such data that can organized into two main categories: static and dynamic. Static data includes static production information (such as products, product families, setup times, processing times) including the current shop-floor configuration (such as production lines, production line types, workplaces, workplace types etc). Dynamic data contains information that can be different between different invocations of the optimization module, such as the production orders that have to be executed as well as scheduled or unscheduled maintenance activities that have to be performed.

In the following experiment, we took a data sample with 10 production orders. These data include maintenance activities as well as resource requirements to produce each SKU. We experimented with the existence or not of maintenance activities as well as with the replenishment quantities. We also assumed 3 replenishment policies per resource:

- Policy 1: 1000 units / hour
- Policy 2: 2000 units / hour
- Policy 3: 4000 units / hour

Below, Figures 15 to 28 capture the resource availabilities over time and the Gannt charts for all combinations of replenishment policies 1, 2 and 3 as well as with and without maintenance activities.

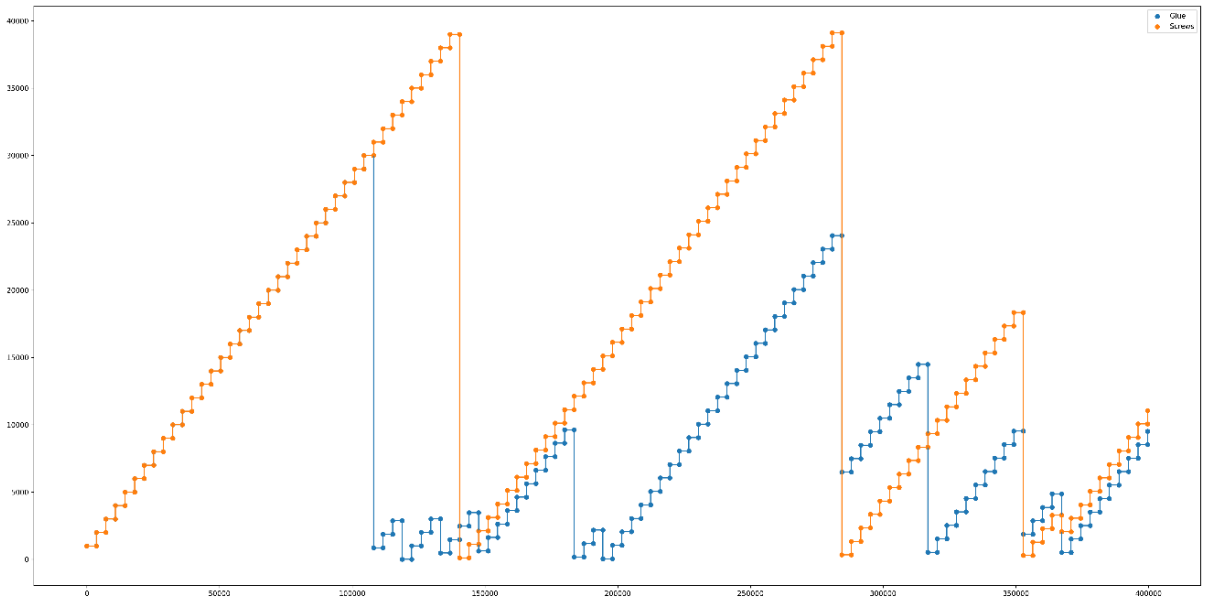

Figure 15: Resource availability over time for Policy 1 without maintenance activities

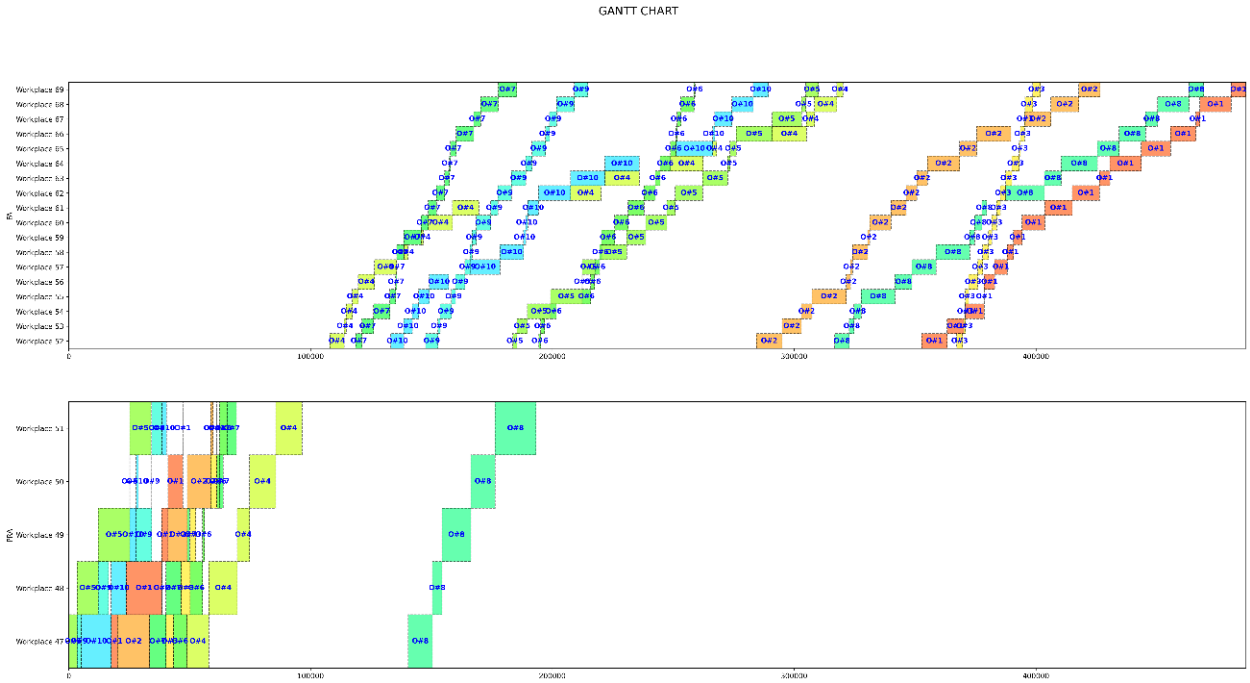

Figure 16: Gantt Chart considering the resource replenishment according to Policy 1 without maintenance activities

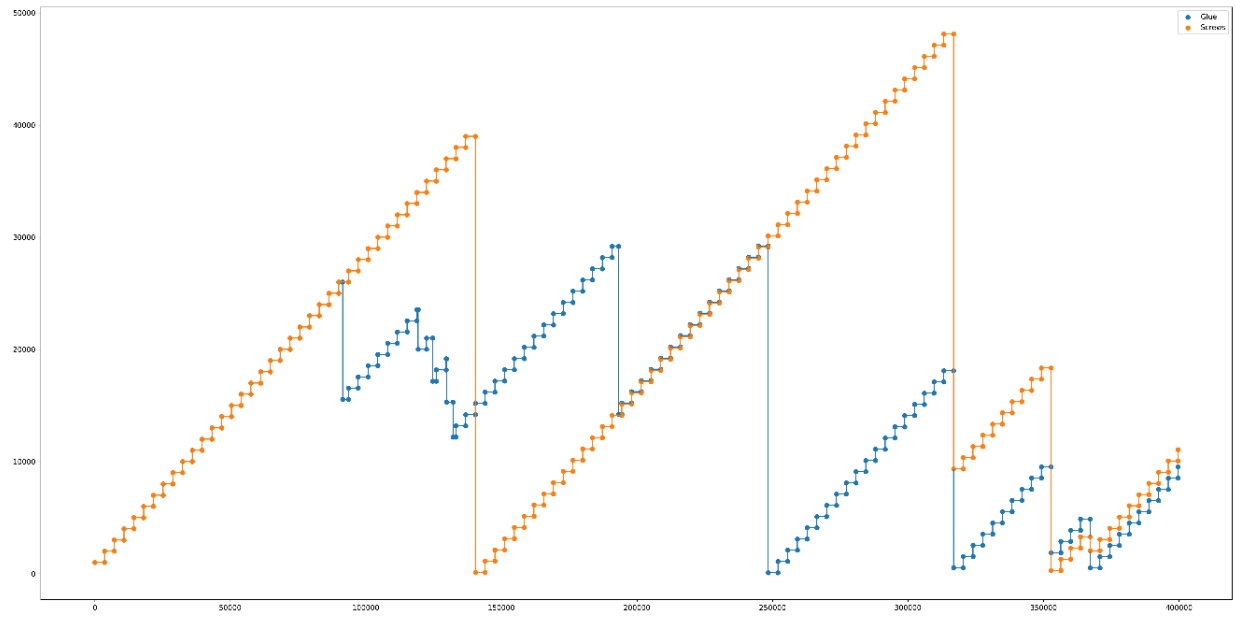

Figure 17: Resource availability over time for Policy 1 with maintenance activities

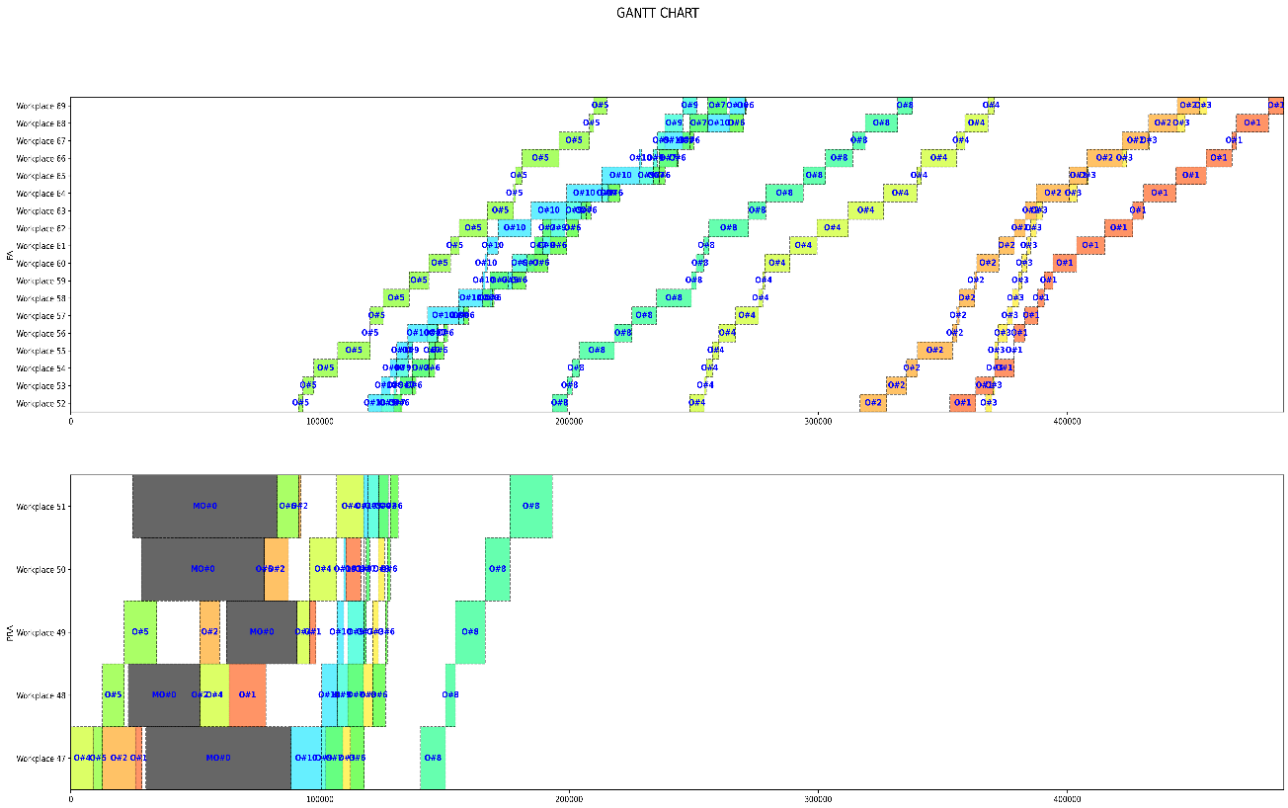

Figure 18: Gantt Chart considering the resource replenishment according to Policy 1 with maintenance activities

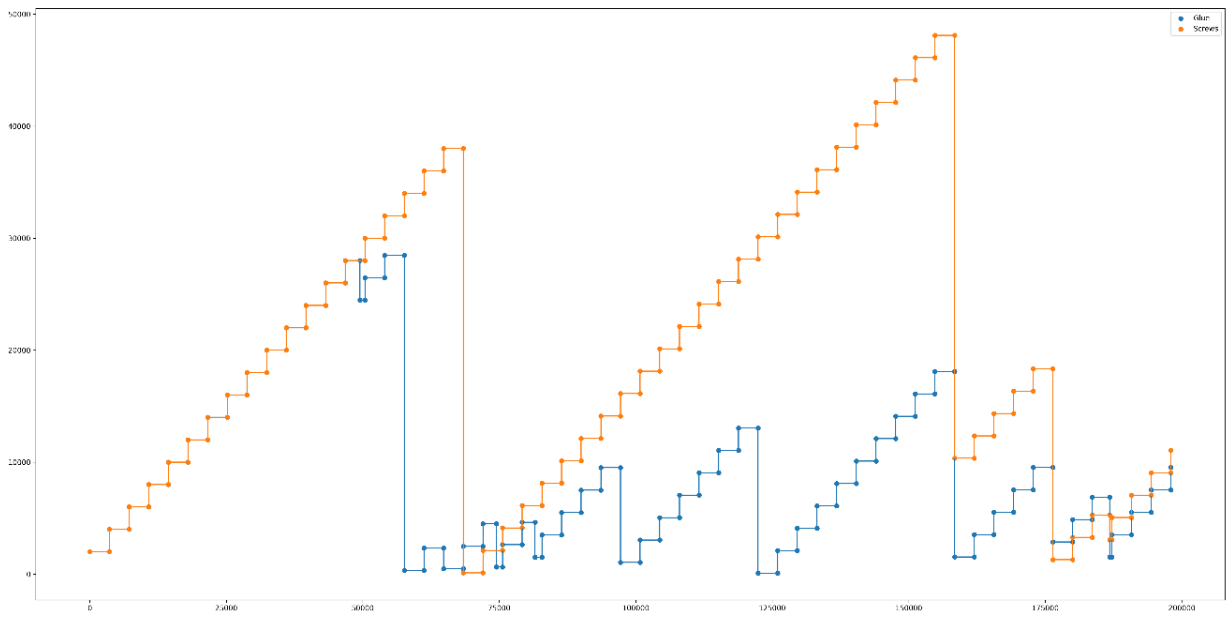

Figure 19: Resource availability over time for Policy 2 without maintenance activities

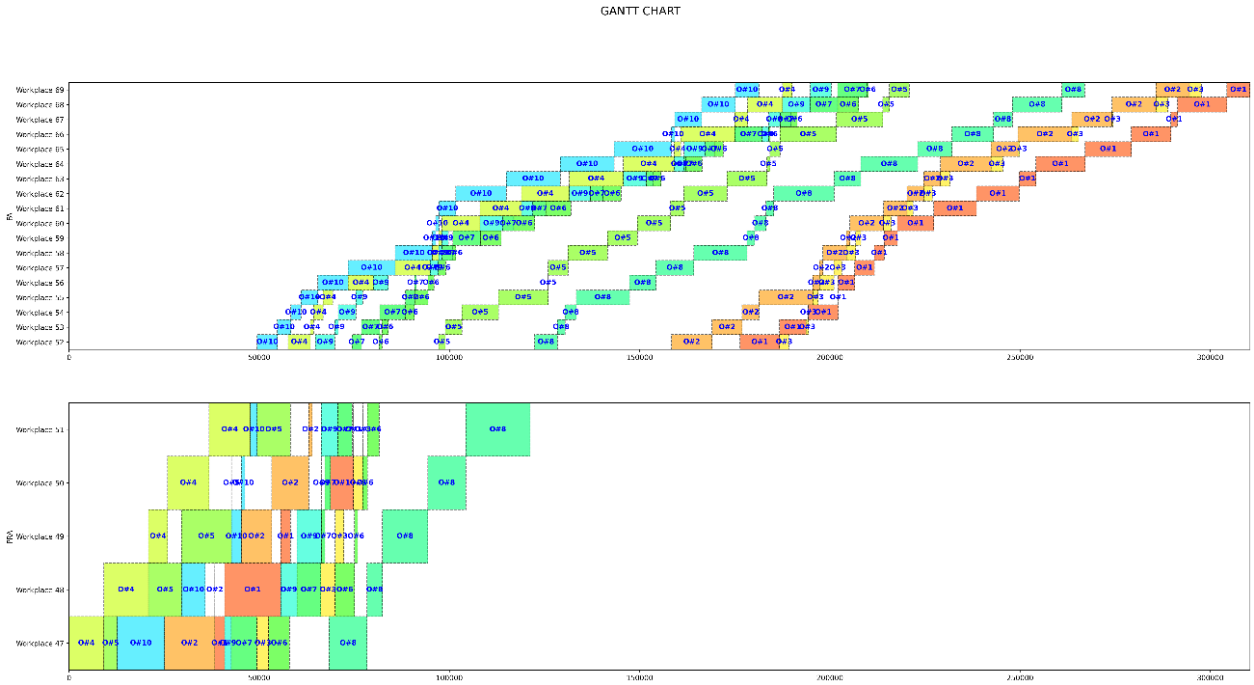

Figure 20: Gantt Chart considering the resource replenishment according to Policy 2 without maintenance activities

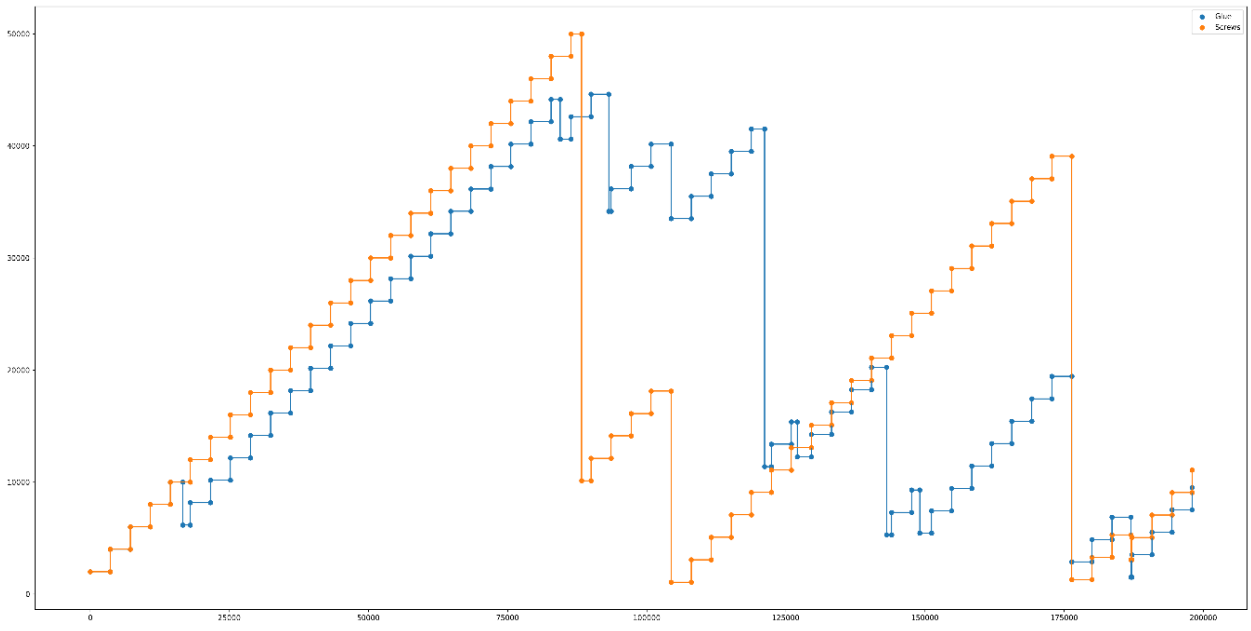

Figure 21: Resource availability over time for Policy 2 with maintenance activities

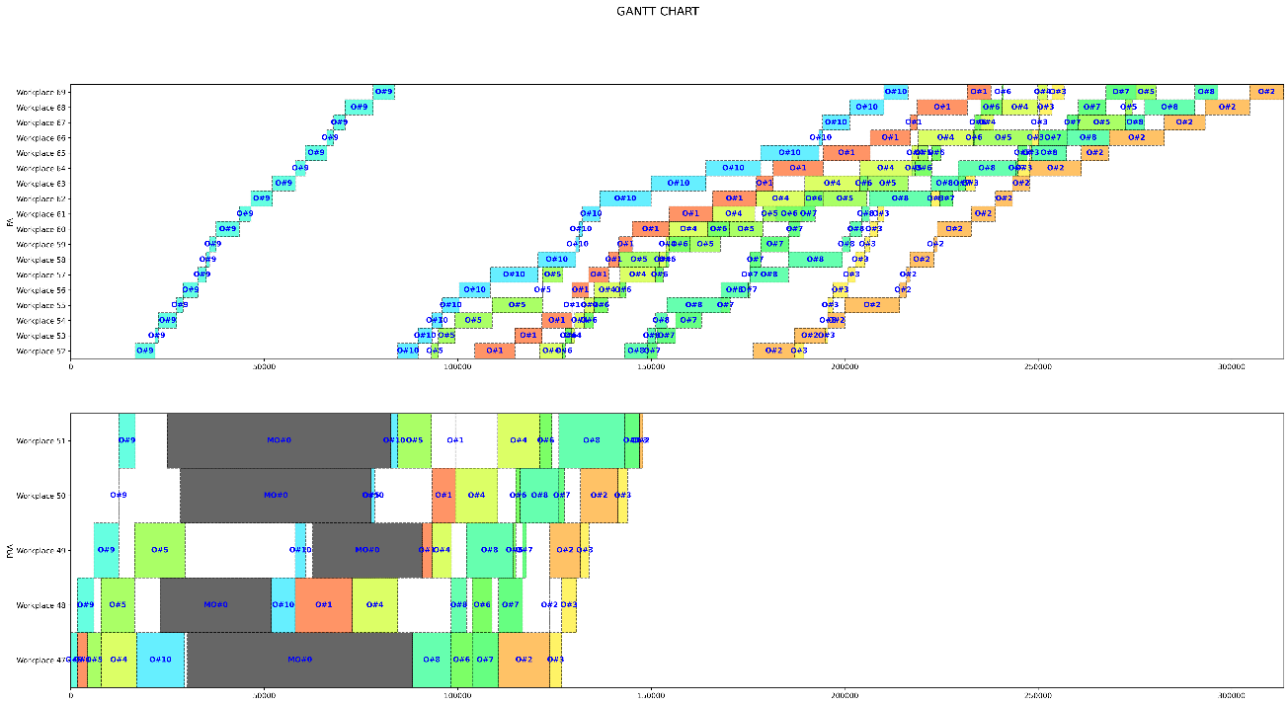

Figure 22: Gantt Chart considering the resource replenishment according to Policy 2 with maintenance activities

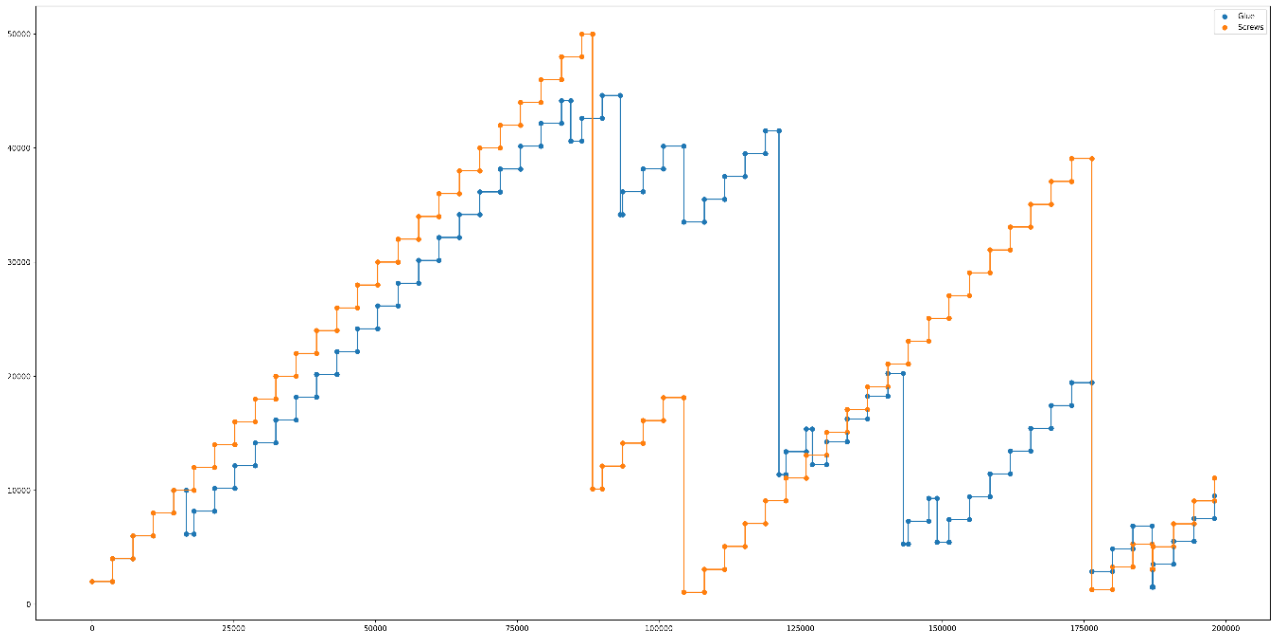

Figure 23: Resource availability over time for Policy 2 with maintenance activities

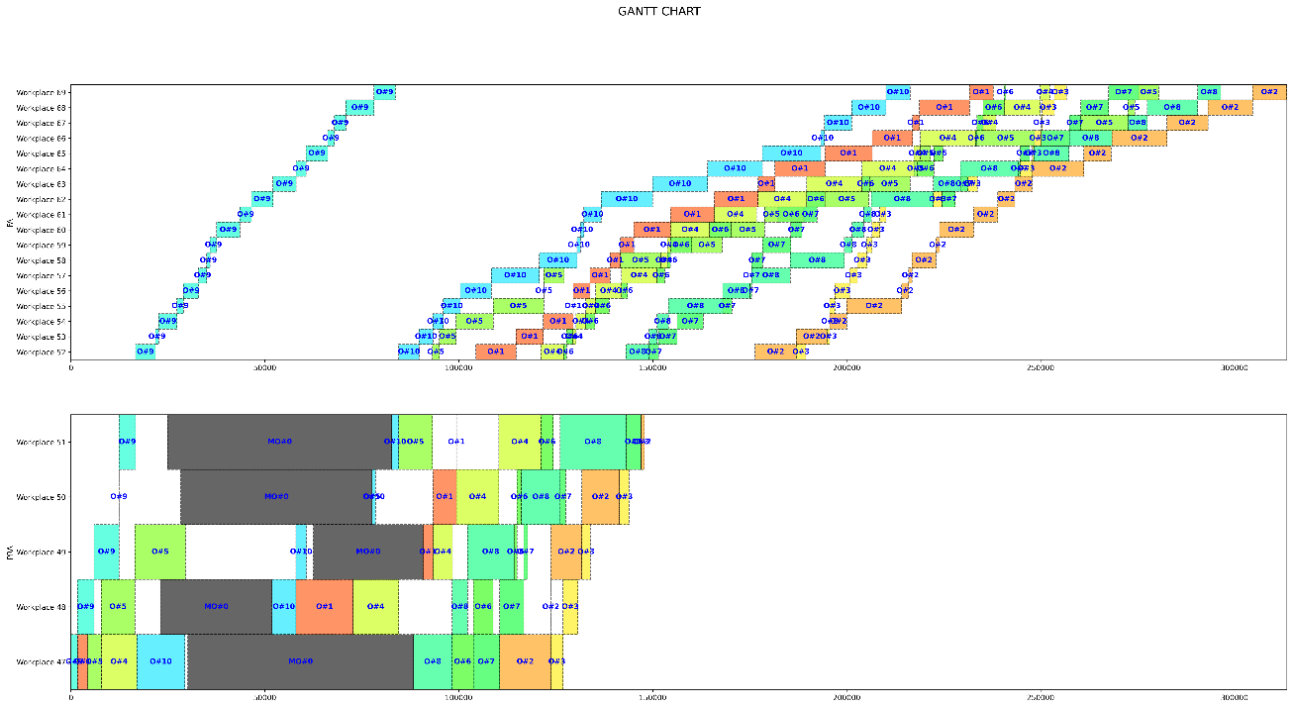

Figure 24: Gantt Chart considering the resource replenishment according to Policy 2 with maintenance activities

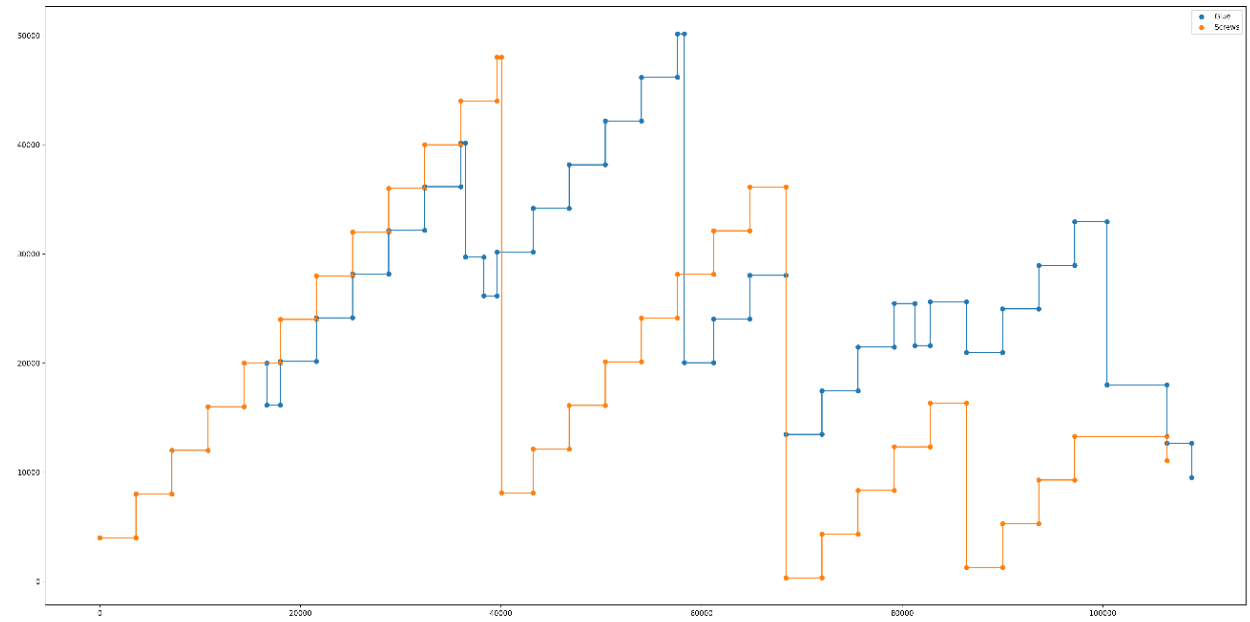

Figure 25: Resource availability over time for Policy 3 without maintenance activities

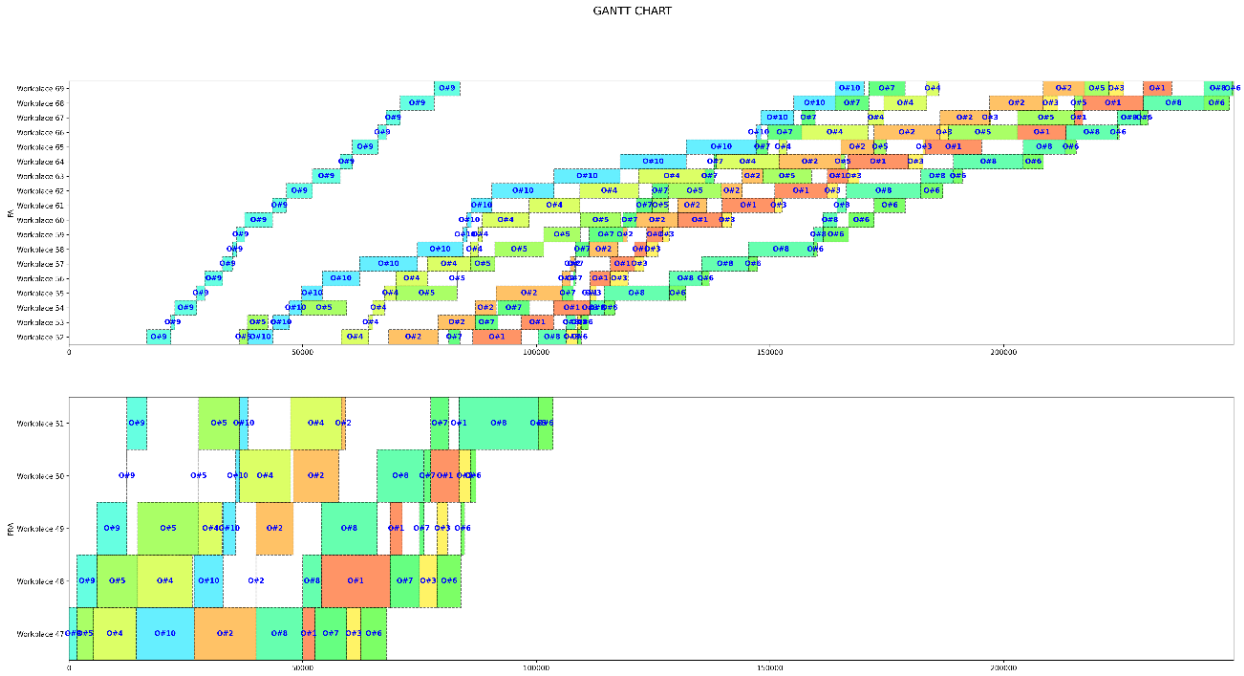

Figure 26: Gantt Chart considering the resource replenishment according to Policy 3 without maintenance activities

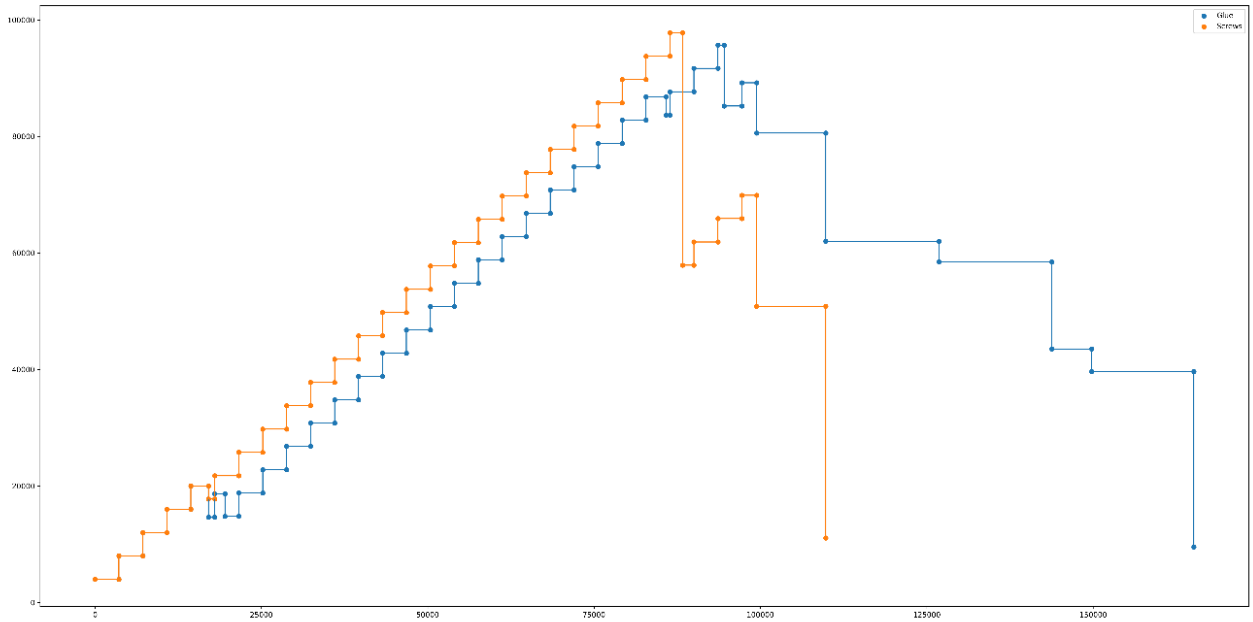

Figure 27: Resource availability over time for Policy 3 with maintenance activities

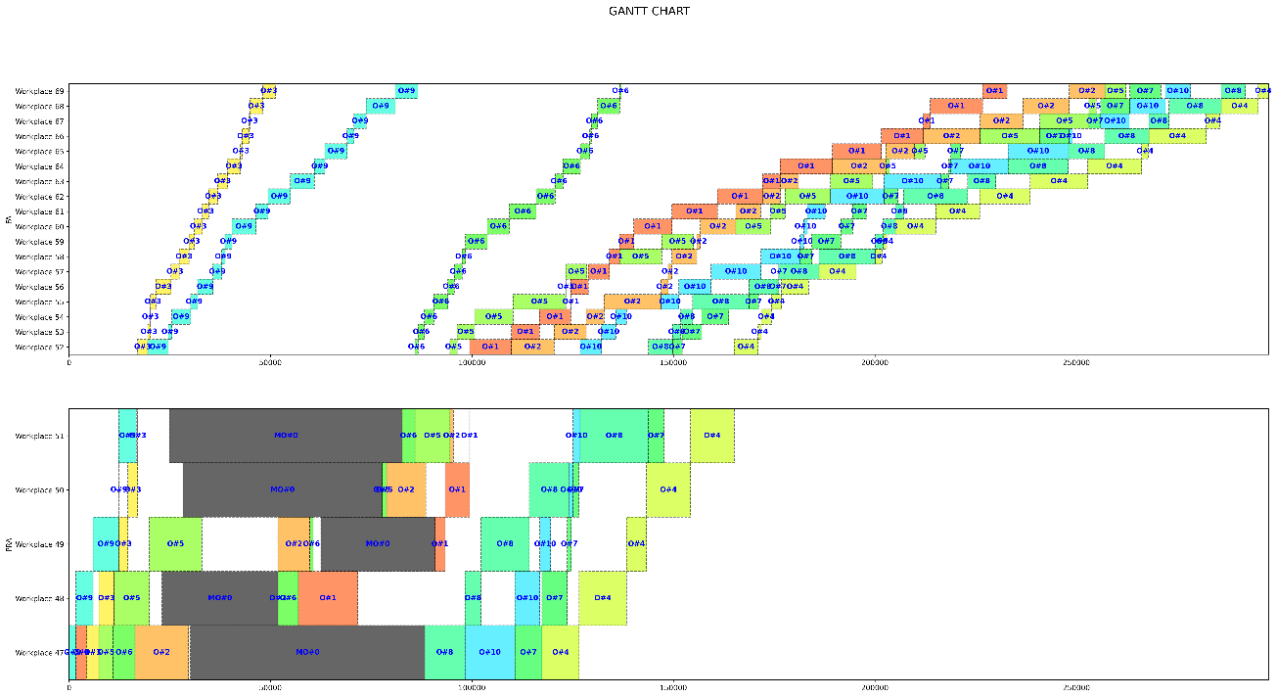

Figure 28: Gantt Chart considering the resource replenishment according to Policy 3 with maintenance activities

The main conclusions that can be drawn from the above experiments are the following:

- Low replenishment rates cause delays in production due to very frequent resource unavailability (see Figures 15 and 17 for Policy 1)
- Higher replenishment rates allow for continuous production without disruptions due to raw material shortage (see Figures 25 and 27 for Policy 3)
- Maintenance activities can cause significant delays, while the impact of maintenance is more evident as production throughput is higher (see Figures 26 and 28).

Table 19 summarizes the results obtained for each case in terms of Makespan and throughput. An increase up to 19.42% when the highest replenishment policy is considered. This indicates that resource constraints and scheduling of maintenance activities can cause significant efficiency losses on production schedules. Therefore, it is very important to invest towards integrated models and schedule simultaneously both product orders and maintenance activities.

	Policy 1		**Policy 2**		**Policy 3**	
#	C_{max}	Throughput	C_{max}	Throughput	C_{max}	Throughput
w. Maintenance	486833	0.85/min	313486	1.31/min	297678	1.38/min
w/o Maintenance	486833	0.85/min	310433	1.33/min	249256	1.65/min

Maintenance Impact	0%	0.97%	19.42%

Table 19: Impact of maintenance activities for different replenishment policies

The above results have been presented at the EURO conference

Repoussis P.P., Kasapidis G., Eirinakis P. and Mourtos Y. (2021). Combined Production Scheduling and Predictive Maintenance for PCB Manufacturing. 31th European Conference on Operational Research conference – EURO 2021, June 11-14, Athens, Greece.

In a final set of computational experiment, we tried to assess the impact of line maintenance activities with respect to the completion time of the schedule. Line maintenance activities block simultaneously the use of all workstations of a production line. We focus on the same set of production orders (see previous set of computational experiments); however, in this case resource constraints are not considered, and we investigate three different scheduled maintenance activity events:

- Event 1: A single pre-assembly maintenance activity
- Event 2: Two maintenance activities on the pre-assembly line
- Event 3: Two maintenance activities on the pre-assembly line and one maintenance activity on the final assembly line.

Figure 29 shows the Gantt chart of the production schedule when no maintenance events are considered, while Figures 30 to 32 show the resulting Gantt charts considering maintenance Events 1 to 3 respectively. One can observe the large idle times that are created on the lines when maintenance takes place. This is more evident when final assembly is maintained that contains more workstations.

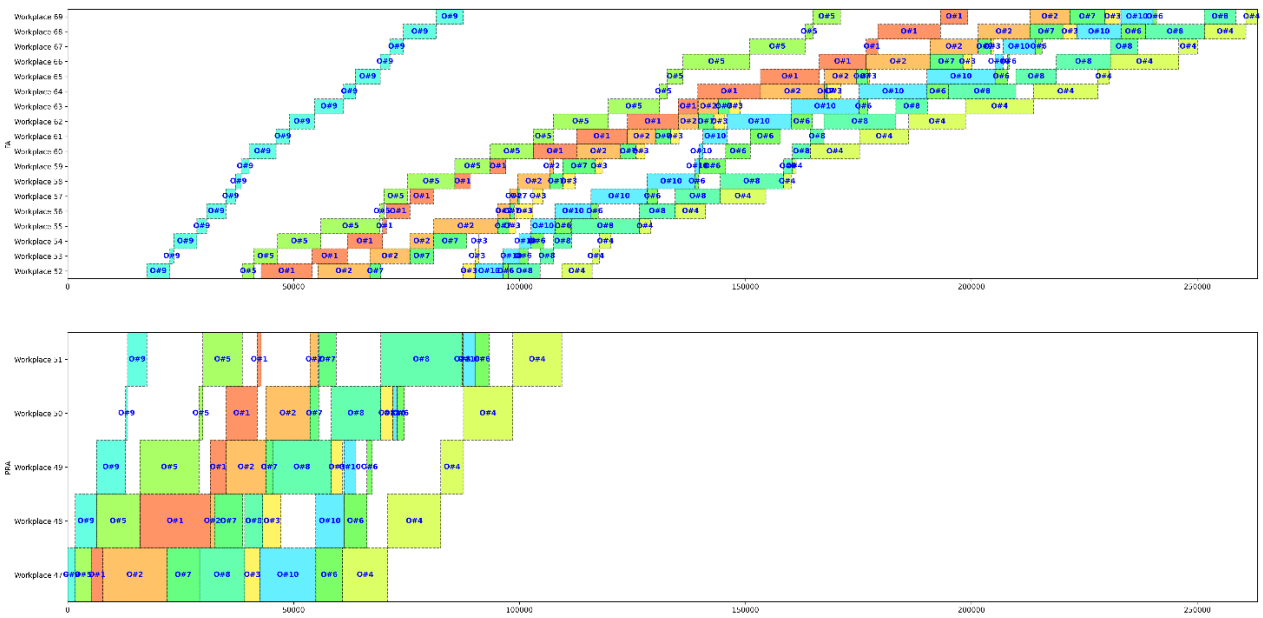

Figure 29: Gantt Chart of the production schedule without maintenance activities

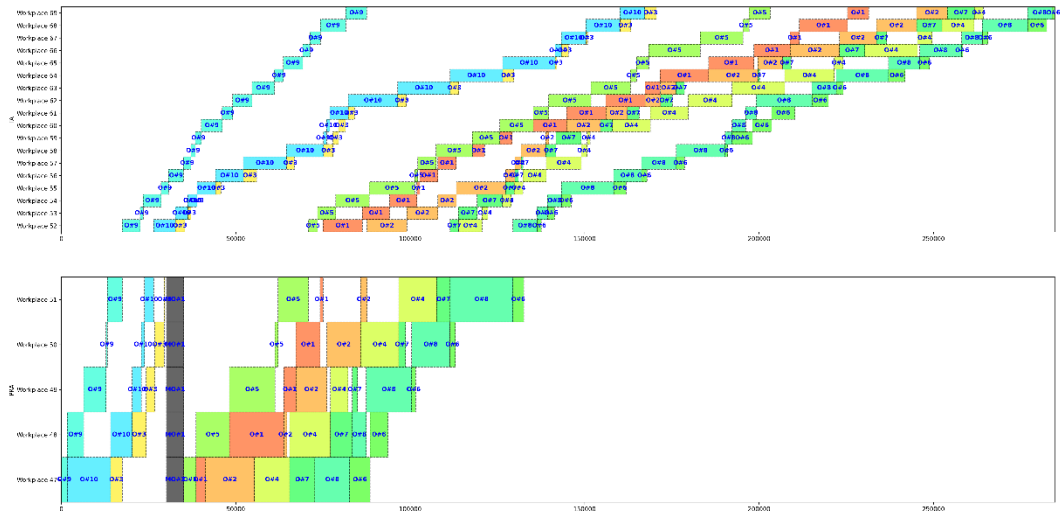

Figure 30: Gantt Chart of the production schedule based on maintenance event 1

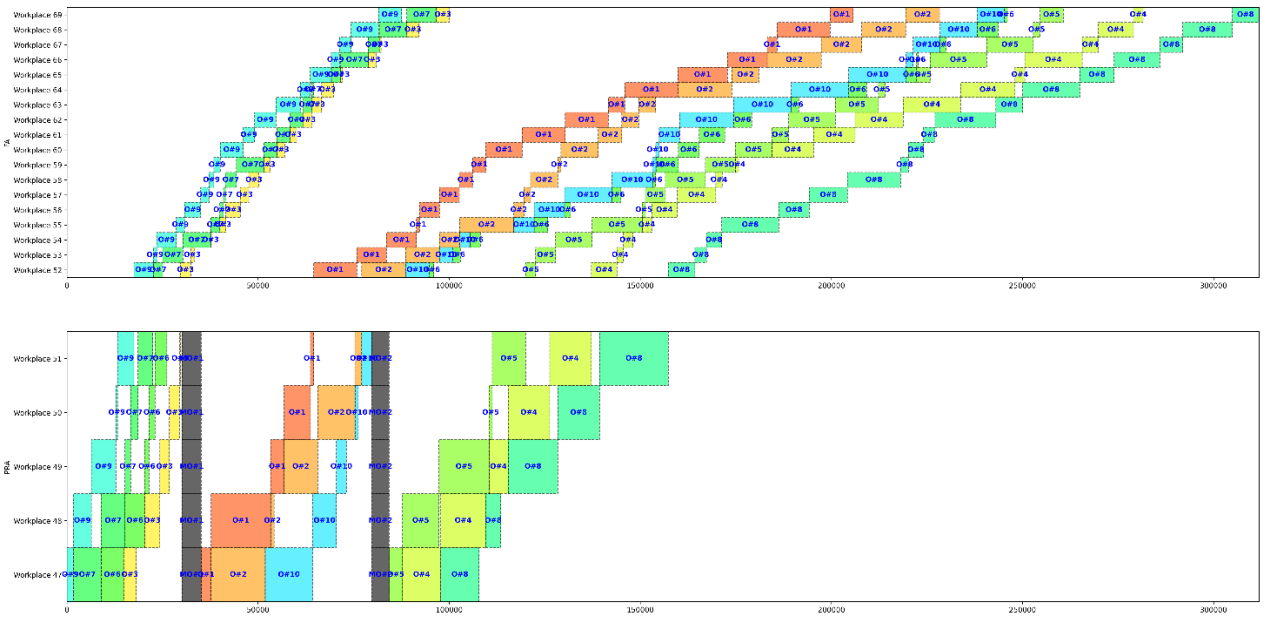

Figure 31: Gantt Chart of the production schedule based on maintenance event 2

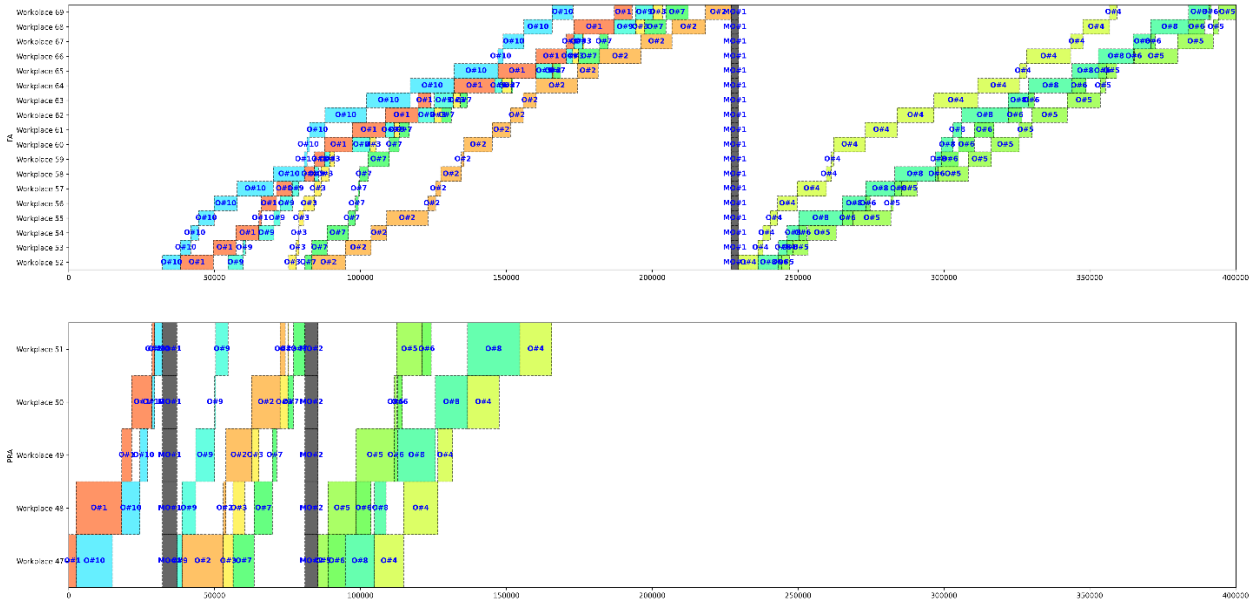

Figure 32: Gantt Chart of the production schedule based on maintenance event 3

Table 20shows the makespan and the throughput of the generated schedules. We notice a staggering 51.99% increase of the makespan in the case of Event 3, where both lines must be maintained. The large number of workstations as well as the necessary blocking of the entire line for the completion of this type of maintenance activity are the root causes of this result. This highlights the need to adopt integrated production and maintenance planning and scheduling tools.

	No Event		**Event 1**		**Event 2**		**Event 3**	
#	C_{max}	Throughput	C_{max}	Throughput	C_{max}	Throughput	C_{max}	Throughput
	263284	1.56/min	284752	1.45/min	311871	1.32/min	400174	1.03/min
Maintenance Impact	0%		8.15%		18.45%		51.99%	

Table 20: Impact of different line maintenance events on the completion time of the schedule

7. References

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.

Al-Qahtani, K., and Elkamel, A. (2008). Multisite facility network integration design and coordination: An application to the refining industry. Computers & Chemical Engineering, 32(10), 2189-2202.

Al-Qahtani, K., and Elkamel, A. (2010). Robust planning of multisite refinery networks: Optimization under uncertainty. Computers & chemical engineering, 34(6), 985-995.

Albahri, T. A., Khor, C. S., Elsholkami, M., and 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.

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

Arabi, M., Yaghoubi, S., and Tajik, J. (2019). Algal biofuel supply chain network design with variable demand under alternative fuel price uncertainty: A case study. Computers & Chemical Engineering, 130, 106528

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.

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.

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

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

Bahadori, Alireza. (2014). Natural gas processing: technology and engineering design. Gulf Professional Publishing.

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

Bektur, G., Sara ̧c, T. A mathematical model and heuristic algorithms for an unrelated parallel machine scheduling problem with sequence-dependent setup times, machine eligibility restrictions and a common server. Computers & Operations Research, 103, 46-63, 2019.

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.

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

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-

Benders, J.F., Partitioning procedures for solving mixed-variables programming problems. Numerische Mathematik, 4, 238-252 (1962).

Birge, J. R. (1997). State-of-the-art-survey—Stochastic programming: Computation and applications. INFORMS journal on computing, 9(2), 111-133.

Birge, J. R., and Louveaux, F. (2011). Introduction to stochastic programming. Springer Science & Business Media.

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

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

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

Brucker, P. Scheduling Algorithms. Springer, 1999

Charnes, A., and Cooper, W. W. (1959). Chance-constrained programming. Management Science, 6(1), 73-79.

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

Charnes, A., Granot, F., & Phillips, F. (1977). An algorithm for solving interval linear programming problems. Operations research, 25(4), 688-695.

Cheng, C.Y., Huang, L.W. Minimizing total earliness and tardiness through unrelated parallel machine scheduling using distributed release time control. Journal of Manufacturing Systems, 42, 1-10, 2017.

Chinneck, J. W., & Ramadan, K. (2000). Linear programming with interval coefficients. Journal of the operational research society, 51(2), 209-220.

Correa, J., Marchetti-Spaccamela, A., Matuschke, J., Stougie, L., Svensson, O., Verdugo, V. , Verschae, J., Strong LP formulations for scheduling splittable jobs on unrelated machines. Mathematical Programming, 154, 305-328 (2015).

Correa, J., Verdugo, V., Verschae, J., Splitting versus setup trade-offs for scheduling to minimize weighted completion time. Operations Research Letters, 44:4, 469-473 (2016).

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

Dalei, N. N., and Joshi, J. M. (2020). Estimating technical efficiency of petroleum refineries using DEA and tobit model: An India perspective. Computers & Chemical Engineering, 142, 107047.

Despotis, D. K., and Smirlis, Y. G. (2002). Data envelopment analysis with imprecise data. European Journal of Operational Research, 140(1), 24-36.

Despotis, D.K. (2005). A reassessment of the human development index via data envelopment analysis. Journal of the Operational Research Society, 56(8), 969-980.

de Almeida Franco, S. V., da Cunha Ribeiro, D., and Meneguelo, A. P. (2020). A comprehensive approach to evaluate feed stream composition effect on natural gas processing unit energy consumption. Journal of Natural Gas Science and Engineering, 83, 103607.

de Gouvea, M. T., and 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.

Dogramaci, A., Surkis, J. Evaluation of a Heuristic for Scheduling Independent Jobs on Parallel Identical Processors. Management Science, 25:12, 1208-1216, 1979.

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

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.

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

Eirinakis, P., Kasapidis, G., Mourtos, I., Repoussis, P., Zampou, E., Situation aware manufacturing systems for capturing and handling disruptions Journal of Manufacturing Systems 58, 365–383

Eroglu, D.Y., Ozmutlu, H.C., Ozmutlu, S., Genetic algorithm with local search for the unrelated parallel machine scheduling problem with sequence-dependent setup times. International Journal of Production Research, 52:19, 5841-5856 (2014).

Eroglu, D.Y., Ozmutlu, H.C., Solution method for a large-scale loom scheduling problem with machine eligibility and splitting property. The Journal of the Textile Institute, 108:12, 2154-2165 (2017).

Fattahi, P., Mehrabad, M.S, Jolai. F. (2007). Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. Journal of Intelligent Manufacturing, 18(3), 331–342.

Fotakis, D., Milis, I., Papadigenopoulos, O., Vassalos, V., Zois, G., Scheduling MapReduce jobs on identical and unrelated processors. Theory of Computing Systems, 64:5, 754- 782 (2016).

Fotakis, D., Matuschke, J., Papadigenopoulos, O., Malleable scheduling beyond 24 identical machines. https: // arxiv. org/ abs/ 1903. 11016 (2020).

Galán-Martín, Á., Guillén-Gosálbez, G., Stamford, L., and Azapagic, A. (2016). Enhanced data envelopment analysis for sustainability assessment: A novel methodology and application to electricity technologies. Computers & Chemical Engineering, 90, 188-200.

Getu, M., Mahadzir, S., and Lee, M. (2012). Analyzing the effects of uncertainties on the economic performance of a chemical process plant using a probabilistic optimization technique. Computer Aided Chemical Engineering, 30, 832-836.

Getu, M., Mahadzir, S., and Lee, M. (2013). Profit optimization for chemical process plant based on a probabilistic approach by incorporating material flow uncertainties. Computers & chemical engineering, 59, 186-196.

Getu, M., Mahadzir, S., Samyudia, Y.,Khan, M.S., Bahadori, A., and Lee, M. (2015) Risk-based optimization for representative natural gas liquid (NGL) recovery processes by considering uncertainty from the plant inlet. Journal of Natural Gas Science and Engineering, 27, 42–54.

Gong, S., Shao, C., and Zhu, L. (2017a). Energy efficiency evaluation in ethylene production process with respect to operation classification. Energy, 118, 1370-1379.

Gong, S., Shao, C., and Zhu, L. (2017b). Energy efficiency evaluation based on DEA integrated factor analysis in ethylene production. Chinese journal of chemical engineering, 25(6), 793-799.

Gonzalez-Garay, A., and Guillen-Gosalbez, G. (2018). SUSCAPE: A framework for the optimal design of SUStainable ChemicAl ProcEsses incorporating data envelopment analysis. Chemical Engineering Research and Design, 137, 246-264.

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.

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

Gurobi Optimization, https: // www. gurobi. com/

Han, Y., and Geng, Z. (2014). Energy efficiency hierarchy evaluation based on data envelopment analysis and its application in a petrochemical process. Chemical Engineering & Technology, 37(12), 2085-2095.

Han, Y.M., Geng, Z.Q., Zhu, Q.X., and Qu, Y.X. (2015). Energy efficiency analysis method based on fuzzy DEA cross-model for ethylene production systems in chemical industry. Energy, 83, 685–695.

Han, Y., Geng, Z., Wang, Z., and 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.

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.

Henrion, R., Li, P., Möller, A., Steinbach, M. C., Wendt, M., and Wozny, G. (2001). Stochastic optimization for operating chemical processes under uncertainty. In Online optimization of large scale systems, 457-478. Springer, Berlin, Heidelberg.

Henrion, R., and Möller, A. (2003). Optimization of a continuous distillation process under random inflow rate. Computers & Mathematics with Applications, 45, 247-262.

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

Hooker J.N., Ottosson, G., Logic-based Benders decomposition Mathematical Programming, 96, 33-60 (2003).

Hooker, J.N., Planning and Scheduling by Logic-Based Benders Decomposition. Operations Research, 55:3, 588-602 (2007).

IBM ILOG CPLEX Optimizer, https: // www. ibm. com/ analytics/ cplex-optimizer.

IBM ILOG CPLEX Optimization Studio CPLEX User's Manual, Version 12 Release 8, Starting from a solution: MIP starts. https: // www. ibm. com/ docs/ en/ SSSA5P_ 12. 8. 0/ ilog. odms. studio. help/ pdf/ usrcplex. pdf, 255-260.

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

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

Kim, J, Kim, H. J., Parallel machine scheduling with multiple processing alternatives and sequence-dependent setup times International Journal of Production Research, 59:18, 5438-5453 (2020).

Kim, J, Kim, H. J., Rescheduling of unrelated parallel machines with job dependent setup times under forecasted machine breakdown International Journal of Production Research, 59:17, 5236-5258 (2020).

Kim, J. Song S., Jeong B., Minimising total tardiness for the identical parallel machine scheduling problem with splitting jobs and sequence-dependent setup times International Journal of Production Research, 58:6, 1628-1643 (2020).

Kim, Y.D., Shim, S.O., Kim, S.B., Choi, Y.C., Yoon, H.M., Parallel machine scheduling considering a job-splitting property. International Journal of Production Research, 42:21, 4531-4546 (2004).

Kim, H.J., Lee, J.H., Scheduling uniform parallel dedicated machines with job splitting, sequence-dependent setup times, and multiple servers. Computers & Operations Research, 126, (2021).

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.

Kuo, T. H., and 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.

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.

Lam, E., Gange, G., Stuckey, P.J., Van Hentenryck, P., Dekker, J.J., Nutmeg: a MIP and CP Hybrid Solver Using Branch-and-Check. SN Operations Research Forum, 1:22, (2020).

Lee, J.H., Hoon Jang, H., Kim, H.J., Iterative job splitting algorithms for parallel machine scheduling with job splitting and setup resource constraints. Journal of the Operational Research Society, 72:4, (2020).

Leiras, A., Ribas, G., Hamacher, S., & Elkamel, A. (2011). Literature review of oil refineries planning under uncertainty. International Journal of Oil, Gas and Coal Technology, 4(2), 156-173.

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

Li, P., Wendt, M., Arellano-Garcia, H., and Wozny, G. (2002). Optimal operation of distillation processes under uncertain inflows accumulated in a feed tank. AIChE Journal, 48(6), 1198-1211.

Li, W., Hui, C. W., Li, P., and Li, A. X. (2004). Refinery planning under uncertainty. Industrial & engineering chemistry research, 43(21), 6742-6755.

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

Li, P., Arellano-Garcia, H., and Wozny, G. (2008). Chance constrained programming approach to process optimization under uncertainty. Computers & Chemical Engineering, 32, 25–45.

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

Maecker, S., Shen, L. Solving parallel machine problems with delivery times and tardiness objectives. Annals of Operations Research, 285, 315-334, 2020. 15

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

Mesfin, G., and Shuhaimi, M. (2010). A chance constrained approach for a gas processing plant with uncertain feed conditions. Computers & chemical engineering, 34(8), 1256-1267.

Mete, E., and Turkay, M. (2018). Energy network optimization in an oil refinery. Computer Aided Chemical Engineering, 44,1897-1902.

Miller, C.E., Tucker, A.W., Zemlin, R.A. Integer Programming Formulation of Traveling Salesman Problems. Journal of the ACM, 7:4, 326-329 (1960).

Miller, B. L., and Wagner, H. M. (1965). Chance constrained programming with joint constraints. Operations Research, 13(6), 930-945

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

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

Moser, M., Musliu, N., Schaerf, A., Winter, F. Exact and metaheuristic ap proaches for unrelated parallel machine scheduling. Journal of Scheduling,315-334, 2020

Mourtos, I., Vatikiotis, S., Zois, G., Scheduling Jobs on Unrelated Machines with Job Splitting and Setup Resource Constraints for Weaving in Textile Manufacturing. APMS 2021: Advances in Production Management Systems. Artificial Intelligence for Sustainable and Resilient Production Systems, 424-434 (2021).

Murali, A., Berrouk, A.S., Dara, S., AlWahedi, Y.F., Adegunju, S., Abdulla, H.S., Das, A.K., Yousif, N. and Hosani, M.A. (2020). Efficiency enhancement of a commercial natural gas

liquid recovery plant: A MINLP optimization analysis. Separation Science and Technology, 55(5), 955-966.

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

Ozturk, O., Chu, C. Exact and metaheuristic algorithms to minimize the total tardiness of cutting tool sharpening operations. Expert Systems with Applications, 95, 224-235, 2018.

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

Peyro, L.F., Models and an exact method for the Unrelated Parallel Machine scheduling problem with setups and resources. Expert Systems with Applications: X, 5, (2020).

Peyro, L.F., Ruiz, R., Perea, F., Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times. Computers & Operations Research, 81, 173-182 (2019).

Pinto, J. M., and Moro, L. F. (2000). A mixed integer model for LPG scheduling. Computer Aided Chemical Engineering, 8, 1141-1146.

Prekopa, A., (1995). Stochastic Programming. Mathematics and Its Applications, vol 324. Springer, Dordrecht.

Qyyum, M. A., Naquash, A., Haider, J., Al-Sobhi, S. A., and Lee, M. (2022). State-of-the-art assessment of natural gas liquids recovery processes: Techno-economic evaluation, policy implications, open issues, and the way forward. Energy, 238, 121684.

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

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

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

Roberti, R., Toth, P., Models and algorithms for the Asymmetric Traveling Salesman Problem: an experimental comparison. EURO Journal on Transportation and Logistics, 1, 113–133 (2012).

Rosales, O.A., Bello, F.A., Alvarez, A., Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. The International Journal of Advanced Manufacturing Technology, 76, 1705–1718 (2015).

Rožanec, J. M., Trajkova, E., Lu, J., Sarantinoudis, N., Arampatzis, G., Eirinakis, P., ... & Mladenić, D. (2021). Cyber-Physical LPG Debutanizer Distillation Columns: Machine-Learning-Based Soft Sensors for Product Quality Monitoring. Applied Sciences, 11(24), 11790.

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

Ruszczynski A., and Shapiro A., (2003). Stochastic Programming. Handbooks in Operations Research and Management Science, Elsevier, 10, 1-64.

Salas, S. D., Contreras-Salas, L., Rubio-Dueñas, P., Chebeir, J., and Romagnoli, J. A. (2021). A multi-objective evolutionary optimization framework for a natural gas liquids recovery unit. Computers & Chemical Engineering, 151, 107363.

Safari, A., and Vesali-Naseh, M. (2019). Design and optimization of hydrodesulfurization process for liquefied petroleum gases. Journal of Cleaner Production, 220, 1255-1264.

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.

Tran, T.T., Araujo, A., Beck, J.C., Decomposition methods for the parallel machine scheduling problem with setups INFORMS Journal on Computing, 28, 83–95 (2016).

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.

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.

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

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

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

Yalaoui, F., Chu C., An efficient heuristic approach for parallel machine scheduling with job splitting and sequence-dependent setup times. IIE Transactions, 35:2, 183-190 (2003).

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.

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

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

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

Zanin, A. C., de Gouvea, M. T., and 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.

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

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