MATHEURISTIC METHODS FOR SOLVING THE MULTI-CALENDAR NAVAL SURFACE SHIP WORK PERIOD PROBLEM

by

Hyojae Kim

Submitted in partial fulfillment of the requirements
for the degree of Master of Applied Science

at

Dalhousie University
Halifax, Nova Scotia
June 2023

Table of Contents

List of Tables

List of Figures

Abstract

This thesis deals with the development of a binary integer programming (BIP) model and novel matheuristics to solve the naval surface ship work period problem (NSWPP) with multi-calendar activities and resources, multiple types of precedence relationships, and timing requirements. As this extended NSWPP is NP-hard, its computation time increases exponentially with the number of variables. The proposed solution approach reduces computation times by using a decomposition matheuristic method to quickly provide near-optimal solutions. The matheuristic method is a sequential multi-step optimization (MSO) using heuristic priority rules to classify the project activities into subgroups which are then optimally scheduled. Schemes are then devised to construct a final solution from the smaller optimal subgroup solutions. Extensive numerical experiments are then conducted using actual ship refit data. The MSO matheuristic is shown to obtain near optimal feasible solutions for large-scale instances of the problem in reasonable computation times.

List of Abbreviations Used

BIP	Binary Integer Program
CNC	Coefficient of Network Complexity
DDT	Disaggregated Discrete-Time
DR	Disjunction Ratio
DT	Discrete-Time
DWC	Duration Weighted Centroid
DWP	Dry-dock Work Period
ES	Earliest Start
ERP	Enterprise Resource Planning
FCT	Flow-based Continuous Time
FF	Finish to Finish relationship
FS	Finish to Start relationship
LP	Linear Program
LS	Latest Start
MILP	Mixed Integer Linear Program
NSWPP	Naval Surface Ship Work period Problem
OOE	On-Off Event
OS	Order Strength
PR	Process Range
RCPSP	Resource-Constrained Project Scheduling Problem
RC	Resource Constrainedness
RF	Resource Factor
RS	Resource Strength
SEE	Start/End Event Based
SF	Start to Finish relationship
SGS	Schedule Generation Scheme
SS	Start to Start relationship
SWP	Short Work Period

Acknowledgements

Many people have helped me during the course of my graduate studies. I am thankful to my family and friends for their love, support, encouragement, and motivation, which helped me pursue this opportunity. I express my gratitude to them for helping me to complete my thesis.

I am really grateful to my supervisors, Dr. Claver Diallo and Dr. Alireza Ghasemi, for their help and guidance. Also, I would like to thank my supervisory committee members who gave warm encouragement and advice even when they were busy.

I am thankful to LCdr Eric Bertrand and Mr. Sanjay Prabhu for starting this NSWPP project. There were many difficult phases during this work, but I was able to count on my peer Yun Yin.

This work was supported by Mitacs and Thales Canada. Thank you for giving me the opportunity to develop myself further.

Chapter 1

Introduction

The naval surface ship work period problem (NSWPP) is a project scheduling problem in which a large number of activities related to ship periodic maintenance, repairs, renewal and engineering changes (EC) activities are planned and executed (Bertrand, 2020). The NSWPP is a variant of the resource constrained project scheduling problem (RCPSP) with specific network characteristics and constraints including limited work periods, activities of varying priority, multi-calendars activities and resources, multiple precedence relationships, and timing constraints.

The RCPSP aims to determine a project schedule that satisfies the resource constraints and precedence relationships between activities. The resource constraints are induced by the limited number of workers, quantity of available tools, processing time available on machines, space limitation, etc. Precedence relationships are predefined operation sequences that need to be respected. The typical RCPSP aims to create a schedule that minimizes the project completion time while respecting the resource capacity and availability (Schwindt, 2015). The RCPSP has been shown to be strongly NP-hard, *i.e.,* the size of the solution space increases exponentially as the number of activities (problem size) increases (Blazewicz et al., 1983). As a result various solution approaches have been derived to tackle this problem in the literature (Boer et al., 1997; Couch, 2016; Zaman et al., 2020).

The NSWPP is a generalization of the RCPSP and thus it is NP-hard (Bertrand, 2020; Prabhu, 2021). Exact and heuristic approaches may be used to solve such problem. However, for large scale problems an exact solution method using commercial solvers may take more time than what a typical schedule planner doing what-if analyses may be prepared to spend. Heuristic methods are generally capable of producing feasible solutions, but there is no guarantee on the quality of the solution obtained.

A recent trend is the combination of mathematical optimization within the scope of construction heuristics to provide higher quality solution. Such methods called matheuristics have yielded promising results (Prabhu, 2021).

Available commercial project management software packages are typically not capable of handling the specific requirements of the NSWPP. As a result a manual or semi-automated project scheduling approach may be used in such environment (Bertrand, 2020). This not only is time consuming, but also may result in inefficient sub-optimal schedule. Also new and emerging jobs may arise due to inspections or other repair and maintenance activities which calls for multiple project re-scheduling during the project execution time. For these reasons, the need for efficient, fast and accurate automated project scheduling systems for naval surface ship work is essential (Bertrand, 2020). This thesis continues the work on the NSWPP started a few year ago.

1.1 Naval Surface Ship Maintenance Operation Scheduling

A typical maintenance lifecycle of naval surface ships is composed of several short work periods (SWP) interspersed with a long dry- dock work period (DWP) flanked by two extended readiness periods (EWP). The naval surface ship work period can increase due to degradation in the condition of the ship or type of the work period. Figure 1.1 shows the naval surface ship maintenance process consisting of three work periods: SWP, EWP and DWP. A ship can have two to four short work periods (SWPs) in a year. Each can take between 12 to 20 weeks to complete depending on scope of work. A DWP, happening almost every 5 years, may take from 20 weeks to 50 weeks depending on the condition and the age of ships (Bertrand, 2020). Before and after each DWP, naval surface ships undergo two extended work periods, each lasting a few months in order to prepare for and after the DWP.

In the NSWPP, multiple work packages composed of several small to large-scale projects have to be scheduled. Tasks in different work packages compete for resources available for the project. The work packages can be generated by three sources:

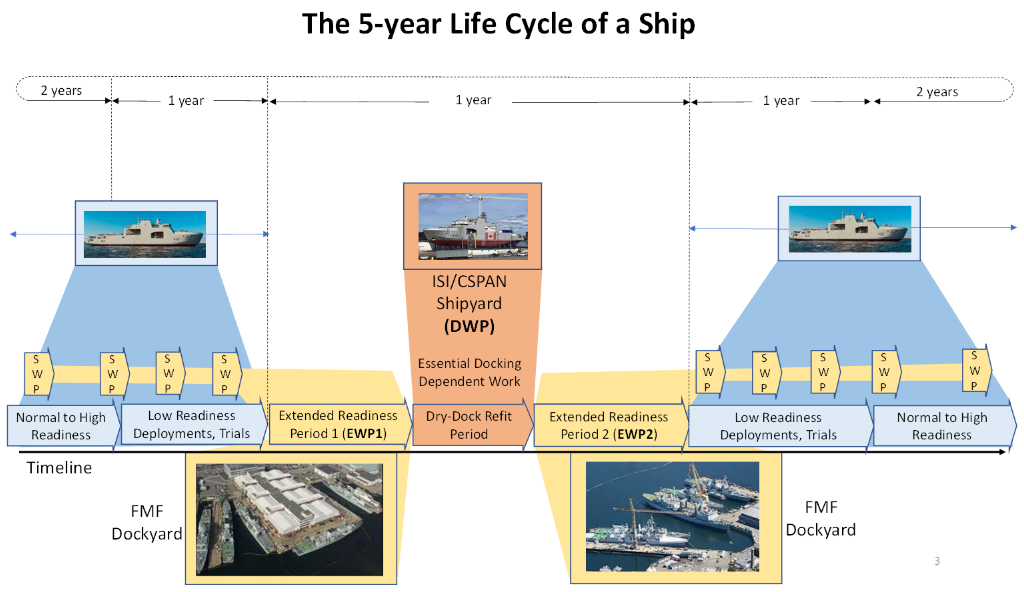
The 5-year Life Cycle of a Ship
2 years
1 year
1 year
1 year
2 years
ISI/CSPAN Shipyard (DWP)
Essential Docking Dependent Work
S W P
Normal to High Readiness
Low Readiness Deployments, Trials
Extended Readiness Period 1 (EWP1)
Dry-Dock Refit Period
Extended Readiness Period 2 (EWP2)
Low Readiness Deployments, Trials
Normal to High Readiness
Timeline
FMF Dockyard
FMF Dockyard
3

Figure 1.1: Naval surface ship maintenance lifecycle (Bertrand, 2020)

preventive maintenance schedules, inspections, and failures noticed during missions. Schedules for the maintenance work periods are then generated using these work packages, associated activities, and all other information needed. The current work package generation process can be seen in Figure 1.2.

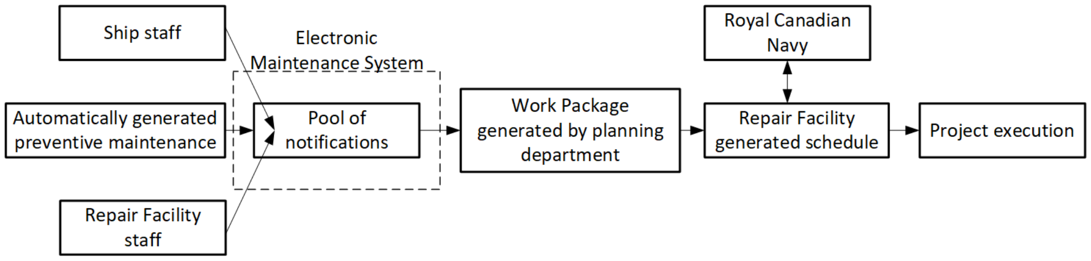
Ship staff
Automatically generated preventive maintenance
Repair Facility staff
Electronic Maintenance System
Pool of notifications
Work Package generated by planning department
Royal Canadian Navy
Repair Facility generated schedule
Project execution

Figure 1.2: Naval ship refit work package and project generation process (Prabhu, 2021)

A planner/scheduler tries to include as many activities as possible in a work period. However, several activities may remain pending or incomplete at the end of the work period. A naval vessel can sail if all essential maintenance works are completed by the end of the work period. Any remaining non essential maintenance work will be planned for a subsequent work period.

1.2 Problem Description and Requirements

The naval surface ship work period problem (NSWPP) is a highly complex scheduling problem with many activities and resource limitations due to activity order, relationships, dependencies and timing constraints. This section briefly reviews key characteristics of the multi-calendar NSWPP considered in this thesis.

1.2.1 Work Packages Priority

The objective of the NSWPP is to complete within the assigned work period all activities required for the ship to safely perform its following mission(s). Not all activities need completing. Typically, three levels of priority are considered for each work package based on the impact of the work package completion on the operation of the ship. Work packages can be labeled *Essential*, *High priority* and *Normal* opportunity work, indicated by priority number 1, 2, and 3 respectively. An essential work package (priority 1) is critical to the successful completion of the next mission and must be completed by the end of the work period. Work packages of priority 2 (High priority) are of less importance to the ship future mission, but also should be complected before the end of the work period. Work packages of priority 3 (Normal) are improvement opportunities and the ship can sail even if all or some of them are not completed by the end of the work period Bertrand (2020). Activities in packages inherit the priority of the corresponding work package. All activities in an essential work package will have priority 1.

1.2.2 Long Duration

Activities in a work package vary in duration from short to long. If a long duration activity is scheduled to be performed near the end of work period, there is a non-negligible probability that it might not be completed in this work period and require postponement to the next maintenance work period. This may mean that the work package has to wait years to be rescheduled again, or if the activity is of priority 1 or 2, it could delay the work period unexpectedly causing unwanted consequences. As a result, it is essential to schedule long duration work packages as early as possible while considering their priority as well. This is called *activity front-loading.*

1.2.3 Multi Calendar Activities and Resources

Activities in a NSWPP may follow different calendars while competing over the same resources. Calendars may be classified using three characteristics: work days in the week, working hours in a day, and holidays. Calendars with 5 or 7 days per week are two examples. For an activity in a 5-day calendar, work cannot take place during the weekends, while a 7-day calendar permits working on every day of the week. Calendars may also have different working hours a day, which in turn will impact the time an activity takes to be completed. For example, some activities follow an 8-hour day calendar while others follow a 12-hour day calendar. An activity with a duration of 24 hours will be completed in 3 days on a 8-hour per day calendar, while it will take 2 days in a 12-hour per day calendar. Holidays exceptions identify days of the year apart from weekends when work cannot be done. Examples are federal statuary or provincial holidays. Resources may also follow different calendars. For example, some equipment may be available on a 7-day per week calendar, while some operators may follow a 5-day per week calendar.

1.2.4 Precedence Constraints

In earlier research, mainly only the Finish-to-Start (FS) precedence relationship between activities in a work package or project were considered Bertrand (2020) and Prabhu (2021). A predecessor activity is an activity that logically precedes a successor activity in the schedule. In this research, three additional precedence relationships between activities are considered. These are Finish-to-Finish (FF), Start-to-Start (SS), and Start-to-Finish (SF).

In the FS precedence relationship, the preceding activity must be finished before the successor activity begins, as shown in Figure 1.3(a). The SF precedence relationship requires that the preceding activity must start before the successor activity is finished 1.3 (b). In the presence of FF precedence relationship, the preceding activity must be finished before the successor activity is finished, as shown in Figure 1.3(c). SS precedence relationship makes sure that the preceding activity starts before the successor activity begins, as shown in Figure 1.3 (d).

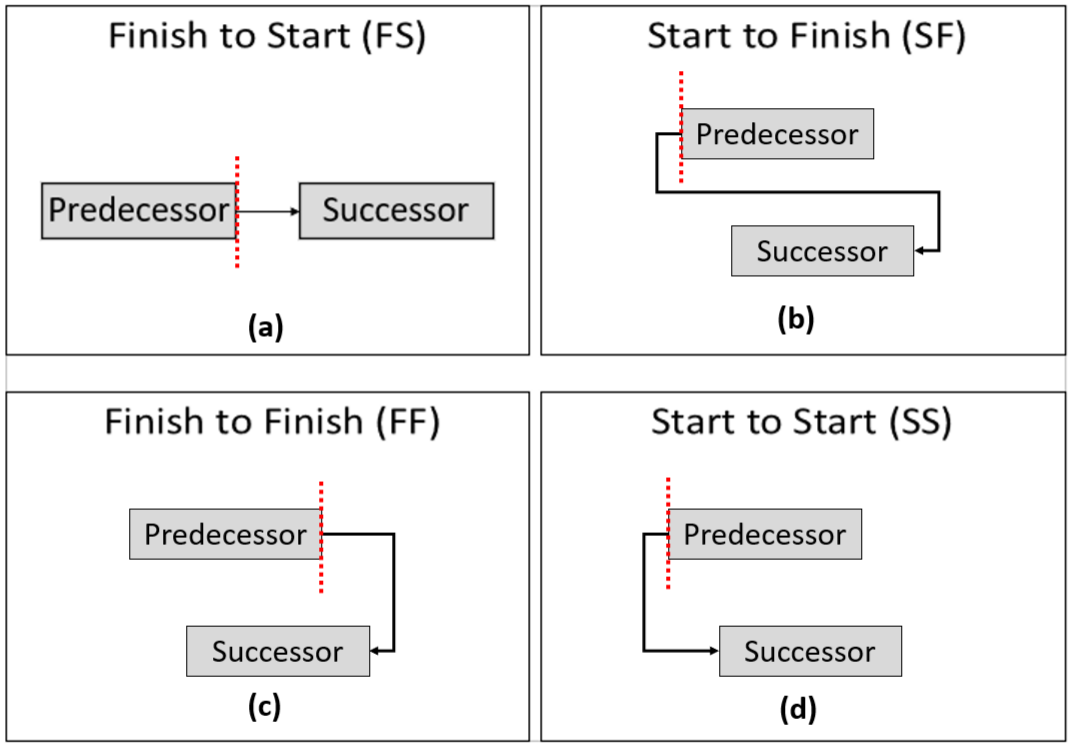

Figure 1.3: Precedence relationships

1.2.5 Start/Finish Time Constraint

Specific time or timing constraints on start and/or finish time of some activities may be required due to some operational requirements. For example, a special tool may be needed to perform tests before a specific activity can start. If the tool is only available on a certain day of the work period, the start of that activity will be limited to the day when the tool is available. There are six types of activity start/finish time constraints: start on, finish-on, start-on-or-after, finish-on-or-after, start-on-or-before, and finish-on-or-before.

An activity with Start-On constraint must start only on the specified date. An activity with Finish-on constraint must be completed on the specified date. Start-on-or-before constraint impose the activity to start-on-or-before a specific date. Start-on-or-after constraints impose the activity to start on or after a specific date. Finish-on-or-before constraint makes sure an activity finishes on or before the specified date. Similarly, Finish-on-or-after constraint, makes sure an activity finishes on or after the specified date.

The term “timing constraints” is used throughout the remainder of this thesis to refer to Start-On, Start-On or Before, Start-On or After, Finish-On, Finish-On or Before, and Finish-On or After constraints.

1.2.6 Resource Constraint

Resources in the NSWPP can be categorized into two types: renewable and non-renewable resources. Renewable resources are those that can be replenished or reused during the project execution, such as human resources or certain types of equipment. Non-renewable resources are those that are limited in supply and cannot be replenished or reused during the project execution, such as spare parts or certain types of specialized equipment.

Each resource in a NSWPP may have a specific calendar associated with it, indicating the availability of the resource over time. The availability of resources is subject to constraints, which must be taken into account when scheduling project activities. These constraints are often related to the limited availability of non-renewable resources or the availability of human resources, which may be required to work on multiple projects simultaneously.

1.2.7 Special Network structure

In the general RCPSP, activities may have multiple precedence relationships to other project activities forming a single project with one critical path while minimizing the total makespan. Figure 1.4 shows an RCPSP Network structure, where the nodes are the activities and the arrows show the precedence relationships between the activities. However, in the NSWPP, work packages have sparse precedence relationship between them, while the activities of a work package are constrained by precedence relationships. All activities in work packagers compete for the same resources (Bertrand, 2020). Figure 1.5 shows an NSWPP Network structure, where the lines show the precedence relationships between activities in a work package and between work packages. NSWPP, unlike RCPSP, does not require all work packages to be completed within a work period, so dummy starting and ending activities (nodes) are not required.

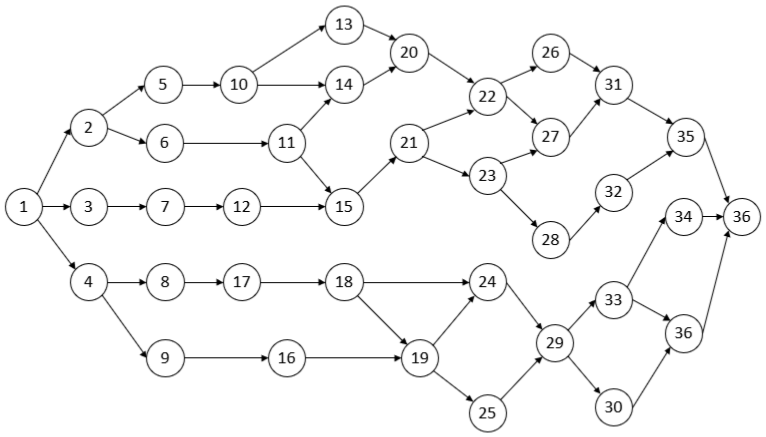

Figure 1.4: Example of a RCPSP project network showing its structure

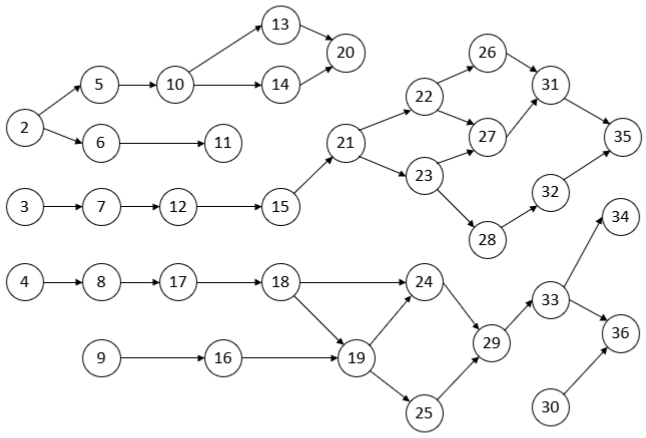

Figure 1.5: Example of a NSWPP project showing its structure

1.3 Thesis Objectives and Outlines

The NSWPP is a variant of the resource constrained project scheduling problem (RCPSP) with its own specific network characteristics and constraints including limited work periods, activities of varying priority, multi-calendar activities and resources, precedence relationships, and time constraints. The objectives of this thesis is to investigate, model and solve the NSWPP to minimize the total weighted priority-duration of completed activities during planning horizon.

The remainder of this thesis is organized as follows. A literature review relevant to project scheduling and RCPSP, and NSWPP is presented in Chapter 2. Chapter 3 will provide a detailed BIP and a matheuristic algorithm to solve the problem in hand. Chapter 4 present numerical examples and test results of Matheuristic algorithm and the serial SGS . Finally, conclusions and future research developments are discussed in Chapter 5.

Chapter 2

Literature Review

The NSWPP is a highly complex variant of the RCPSP, with hundreds of activities to be scheduled using the same shared resource pool. This section provides an overview of RCPSP and NSWPP, including relevant RCPSP variants, current models, solving methods, and research relevant to the multi-calendar NSWPP.

A project consists of a series of activities that take time to execute, require resources, and incur costs or generate cash flow (Schwindt, 2015). There are many factors that can complicate the project schedule impacting activities' scheduled time, resources, and performance. Examples are deadline penalties, priority requirements, and multiple execution modes. Project management involves the coordination of all these characteristics from beginning to end (Sathi et al., 1985). Project Scheduling is the planning process that aims to successfully complete the project. Project scheduling incorporates wide range of input data like project network structure, activity duration, resource requirements, aggregating or dis-aggregating the activities (Kerzner, 2017). There have been many variations of this problem proposed for a variety of applications involving the scheduling of projects in practice (Schwindt, 2015).

Early project scheduling modeling works developed the Critical Path Method (CPM) and Project Evaluation and Review Technique (PERT) in which project activities are scheduled only by considering activities' precedence relationship (J. J. E. Kelley & Walker, 1959; Malcolm et al., 1959). Although project scheduling techniques can make great contribution to efficient project management, their application in real world problems are limited as they fail to properly integrate resources availability and other real-life requirements, which is the main concern of project management.

Resources required to perform projects are often limited, so that even if the activities are ready to start after satisfaction of precedence relationships, they may still incur delays until all needed resources are available. Resource Constrained Project Scheduling Problem (RCPSP) is a project planning and scheduling problem which may consider multiple set of constraints, precedence relationships, resource types and their availability, activity splitting, and execution modes, first introduced by Kelley & E (1963). It has been widely applied and studied for developing or designing large-scale projects, such as software and military industry projects. The RCPSP can be complex and difficult to solve. Indeed, the RCPSP is an NP-hard problem Blazewicz et al. (1983). Therefore, it is impossible or very expensive to find an optimal solution to a problem involving a large number of activities and constraints within a given time. However, it is possible to obtain a feasible solution for a large-scale problem using heuristic methods. Because, solutions obtained by heuristics may be far from the optimal solution, recent advances have seen the recourse to matheuristics to improve solution quality (Kolisch & Hartmann, 2006). The importance of a fast solution approach for RCPSP is even more critical with existence of changes in requirements, and/or need to rescheduling the project.

RCPSPs can be classified according to seven main attributes: type of constraints, type of precedence relations, type of activity splitting, number of execution modes, type of resource, number of objectives, and level of information as detailed in Table 2.1 below.

2.1 Characteristics of NSWPPs

The NSWPP has typically a fixed due date (project horizon), activity duration constraints and/or start time and finish time requirements. No gap is assumed to be required between activities finish or start times. In some problems the resource availability in NSWPPP is further limited as the resources are shared by multiple projects. Renewable resource capacities are limited during each time interval, but will be returned to the pool of the resources when the activity using them is completed. Machines, tools, and labour resources are regarded as renewable resources. Non-renewable resources may also be needed to complete an activity. The non-renewable

Table 2.1: RCPSP Characteristics (Schwindt, 2015)

ATTRIBUTES	**CHARACTERISTICS**
Type of constraints	Time-constrained problem
	Resource-constrained problem
Type of precedence relations	Ordinary precedence relations
	Generalized precedence relations
	Feeding precedence relations
Type of resources	Renewable resources
	Non-renewable resources
	Storage resources
	Continuous resources
	Partially renewable resources
Type of activity splitting	Non-preemptive problem
	Integer preemption problem
	Continuous preemptive problem
Number of execution mode	Single-mode problem
	Multi-mode problem
Number of objective types	Single-criterion problem
	Multi-criteria problem
Level of information	Deterministic problem
	Problem under interval uncertainty
	Stochastic problem
	Problem under vagueness

resources capacity are used by activities over project horizon. NSWPP can be classified as an RCPSP problem with renewable resources. In NSWPP, all activities are assumed to be non-preemptive, meaning jobs are not allowed to be interrupted once they have started. Activities duration may be reduced to respect project due date by selecting execution modes that speed up execution but use more resources.

A typical RSCSP objective is to minimize the makespan of the project. Alternatively, minimizing resource utilization or its deviation may be an objective for the RSCSP optimization. Multi criteria objective function composed of two or more of above mentioned objective functions may also be considered in some problems. In the NWSPP, the objective is to complete as many as activities possible, while front-loading activities with longer duration, and/or higher priority.

Available information and parameters of a scheduling problem, may be deterministic or stochastic (non-deterministic) due to the nature of the problem in the real world. Stochastic information of a project may be presented in terms of uncertainty in the values, modeled by some probabilistic distributions. For example, stochastic information may be applicable to activity duration or resource availability. The NSWPP problem in this thesis is deterministic.

Couch (2016) introduced a robust project scheduling policy through analyzing three buffer placement approaches to improve schedule quality for naval maintenance projects. This study formed the basis for the NSWPP study. Bertrand (2020) developed several binary integer programming (BIP) models for initial scheduling and rescheduling of the NSWPP. Prabhu (2021) developed a decomposition matheuristic to solve large-scale NSWPPs. Prabhu (2021) research included a single calendar and Finish-to-Start precedence relationships only. A recent study by Yin (2022) extended the original NSWPP to include multi-calendar resources and activities, and timing constraints. Yin (2022) introduced a novel universal duration matrix to handle the multi-calendar requirements and a formulation for calculating Earliest Start(ES) and Latest Start(LS) for the additional precedence relationships (FF, SF, and SS).

Table 2.2 summarizes the important characteristics of the NSWPP used by Yin

(2022) and also considered in this thesis.

Table 2.2: NSWPP Characteristics (Yin, 2022)

ATTRIBUTES	**CHARACTERISTICS**
Type of constraints	Time-constrained problem
	Resource-constrained problem
Type of precedence relations	Generalized precedence relations
Type of resources	Renewable resources
Type of activity splitting	Non-preemptive problem
Number of execution mode	Multi-mode problem
Number of objective types	Multi-criteria problem
Level of information	Deterministic problem

2.2 RCPSP Modeling for NSWPPs

The RCPSP model has been extensively studied in the past few decades leading to the development of several linear models and solution techniques. In the following section we will discuss the most important attempts and results in modelling RCPSP and review their characteristics.

Considering an RCPSP with n activities to be completed using K resources. $\mathcal{J} = \{1, \cdots, n\}$ is the set of activities. Activity i has duration d_i and engages r_{ik} units of resource k for its execution. $\mathcal{P}$ is a set of pairs of predecessors (i, j) which means i is the immediate predecessor of j. The project horizon is represented by H.

Pritsker et al. (1969) modeled the RCPSP using a binary decision variables denoting that activity i starts at time t.

$$x_{it} = \begin{cases} 1, & \text{if activity } i \text{ starts at time } t \\ 0, & \text{otherwise.} \end{cases}$$

The decision variable defined creates a binary variable for each activity in each time period. This means that there will be nH binary decision variables. To reduce the number of binary variables, the range of x_{it} can be limited to discrete time periods between the earliest start time ES_i and the latest start time LS_i of activity i. A forward pass and backward pass approach can be performed to determine the earliest start times ES_i and the latest start times LS_i. Equations (2.1) to (2.8) give the formulas to calculate the earliest and latest start times under each of the four types of precedence relationships (*i.e.,* Finish-to-Start (FS), Finish-to-Finish (FF), Start-of-Start (SS), and Start-to-Finish (SF)) when the project start time is set to 0 (Tormos & Lova, 2001).

FS:

$$ES_j = \max[ES_i + d_i]; \quad \forall (i,j) \in \mathcal{P} \tag{2.1}$$

$$LS_i = \min[LS_j - d_i]; \quad \forall (i,j) \in \mathcal{P} \tag{2.2}$$

FF:

$$ES_j = \max[ES_i + d_i - d_j]; \quad \forall (i,j) \in \mathcal{P} \tag{2.3}$$

$$LS_i = \min[LS_j - d_i]; \quad \forall (i,j) \in \mathcal{P} \tag{2.4}$$

SS:

$$ES_j = \max[ES_j]; \quad \forall (i,j) \in \mathcal{P} \tag{2.5}$$

$$LS_i = \min[LS_j]; \quad \forall (i,j) \in \mathcal{P} \tag{2.6}$$

SF:

$$ES_j = ES_i + d_i; \quad \forall (i,j) \in \mathcal{P} \tag{2.7}$$

$$LS_i = \min[LS_j - d_j]; \quad \forall (i,j) \in \mathcal{P} \tag{2.8}$$

The following section presents and briefly discusses commonly used RCPSP models based on Time-indexed, Sequence-Based, and Event-Based formulationss.

The first formulation is the basic discrete-time (DT) formulation with a single activity and time-indexed binary decision variable Pritsker et al. (1969).

Indices:

i	index of activities
j	index of activities
k	index of resource types
t	index of time periods

Parameters for Sets:

n	integer, number of non-dummy activities
H	integer, time horizon considered for scheduling
K	integer, number of resource types

Sets:

J	Set of activities, $J = \{0, \cdots, n\}$ with index i or j
$\mathcal{H}$	Set of time periods, $\mathcal{H} = \{0, \cdots, H\}$ with index t
$\mathcal{K}$	Set of types of resources, $\mathcal{K} = \{1, \cdots, K\}$ with index k
P	Set of finish to start precedence activity pairs (i, j)
W_i	Set of time periods between the early start and late start of activity i, $W_i = \{ES_i, \cdots, LS_i\}$

Other Parameters:

d_i (d_j)	integer, duration of activity i (j)
r_{ik}	integer, activity i demand for resource type k
R_k	integer, maximum availability of resource type k
ES_i	integer, earliest start time of activity i
LS_i	integer, latest start time of activity i
p_i	integer, priority of activity i
θ	exponent used to ponderate the priority of activities, $\theta \geq 0$
ε	very small duration value assigned to activities with no duration (i.e., milestones), $\varepsilon \geq 0$
α	parameter used to give tie-breaking benefits to longer duration activity (α=1.1). α=0 makes the model ignore activity's duration in the objective function ; $\alpha \geq 0$

Discrete-time formulation

$$\text{Minimize} = \sum_{t \in \mathcal{W}_i} t x_{n+1,t} \tag{2.9}$$

subject to:

Activity Completion:

$$\sum_{t \in W_i} x_{it} = 1 \qquad \forall i \in \mathcal{J} \tag{2.10}$$

Predecessor (Finish to Start):

$$\sum_{t \in W_i} t x_{jt} \geq \sum_{t \in W_i} t x_{it} + d_i \qquad \forall (i,j) \in \mathcal{P} \tag{2.11}$$

Resource availability:

$$\sum_{i=1}^{n} r_{ik} \sum_{s=max\{ES_i, t-d_i+1\}}^{min\{LS_i, t\}} x_{is} \leq R_k \qquad \forall t \in \mathcal{H}, \forall i \in \mathcal{J}, \forall k \in \mathcal{K} \tag{2.12}$$

$$x_{it} \in 0, 1 \qquad \forall i \in \mathcal{J}, \forall t \in \mathcal{H} \tag{2.13}$$

The objective function in Equation2.9 minimizes the start time of activity $n+1$, a dummy activity of zero duration created to mark the completion milestone of all activities in the project (ie. makespan). Equations 2.10 ensure that each activity within the project is scheduled exactly once and only once. Equations 2.11 ensure that the finish-to-start constraints are satisfied (i.e., an activity can only start when all its predecessors have been completed). Constraints (2.12) that there are enough of each resource type for all scheduled activities in each time period. Equations (2.13) define the decision variables as binary.

Christofides et al. (1987) proposed a disaggregated discrete time (DDT) formulation that replaced Equation (2.11) with the following equation.

$$\sum_{s=ES_i}^{LS_i} x_{is} + \sum_{s=ES_j}^{min\{LS_j,t+d_i-1\}} x_{is} \leq 1 \qquad \forall(i,j) \in P \tag{2.14}$$

The DDT modeling approach results in a tighter relaxation in the precedence relationship constraint compared to the DT formulation.

Artigues et al. (2003) suggested a flow-based continuous-time formulation (FCT) where activities are represented as flows, and resource constraints are modeled using a continuous time approach. The flow-based representation allows for a more natural and flexible way of modeling resource constraints, which makes FCT models more efficient than other RCPSP models. Artigues et al. (2003) state that the FCT has poor relaxations compared to DT and DDT.

Event-based formulations (Koné et al., 2011, 2013) also have the advantage of dealing with non-integer activity processing times. More importantly, for instances with long scheduling horizons, event-based models involve fewer decision variables in comparison to the models indexed by time. Start/End Event Based Continuous Time Formulation (SEE) and On/Off Event based formulation (OOE) were introduced by Koné et al. (2011). SSE use two types of binary variables to represent the start and end of an event. On/Off Event based formulation (OOE) uses only one type of binary variable per activity. The SEE needs more variables for representing activities than with the OOE. With the SEE, the number of events is exactly equal to the number of activities. The representation of the resource constraints is also simpler and easier Koné et al. (2013).

There is limited applicability of the standard discrete-time RCPSP model alone in the real world of project planning, particularly in industries with fixed deadlines that offer limited incentives to finish early and specially in projects with a large degree of inherent uncertainty. Several resource-constrained project scheduling models applicable to the real world problems are discussed in the following sections.

The standard RCPSP can be extended to include multiple objectives, such as minimizing the overall cost of the project, maximizing the availability of ships, and minimizing the risk of downtime. Multi-objective optimization techniques, such as genetic algorithms and evolutionary algorithms, can be used to find a set of Pareto-optimal solutions that trade-off between the different objectives (Dridi et al., 2012). In naval surface ship maintenance projects, the maintenance activities may have a direct impact on the operational readiness of the ships. This interaction can be modeled by including operational constraints such as availability and mission requirements in the RCPSP (C. Li et al., 2022). Different maintenance strategies, such as preventive maintenance, predictive maintenance, and condition-based maintenance can be considered in the RCPSP. This can be modeled by including different cost and resource usage parameters for different maintenance strategies (Nguyen, 2017). In naval surface ship maintenance projects, the parameters of the RCPSP, such as the duration of activities, resource availability, and costs, can be uncertain. Stochastic programming techniques can be used to model this uncertainty and find solutions that are robust to different scenarios (Couch, 2016). Furthermore, the RCPSP can be extended to include risk management considerations such as the risk of downtime and the risk of over-maintenance. This can be modeled by including risk management metrics such as the probability of failure, in the objective function. The RCPSP can be extended to include the life-cycle costs of the maintenance activities, such as the costs of procurement, operation, maintenance, and disposal. This can be modeled by including life-cycle cost parameters in the objective function (Yang & Frangopol, 2020).

Recently, the standard RCPSP model was extended to cover naval maintenance projects with the goal of front-loading high-priority and long-duration activities (Bertrand, 2020; Prabhu, 2021; Yin, 2022). The priority duration formulation aims to incorporate both the priority and duration information of each activity into the scheduling decision-making process.

Bertrand (2020) introduced a model to maximize the weighted sum of priority-duration activities in a naval refit project. The deterministic and discrete-time MILP formulation explained below is aiming to front-load high-priority and long-duration activities, while multi-modes activities, basic precedence relationship constraints, and resource constraints are available. A limited number of activities are permitted to be executed in overtime in this model.

Indices:

i	index of activities
j	index of activities
t	index of time periods (e.g., days)
m	index of execution modes

Parameters for Sets:

n	integer, number of non-dummy activities
H	integer, time horizon considered for scheduling
K	integer, number of resource types
M	integer, number of execution modes

Sets:

$\mathcal{J}$	Set of activities, $\mathcal{J} = \{0, \cdots, n\}$ with index i or j
$\mathcal{H}$	Set of time periods, $\mathcal{H} = \{0, \cdots, H\}$ with index t
$\mathcal{K}$	Set of types of resources, $\mathcal{K} = \{1, \cdots, K\}$ with index k
$\mathcal{M}$	Set of execution modes, $\mathcal{M} = \{0, \cdots, M\}$ with index m
$\mathcal{P}$	Set of immediate finish-to-start activity pairs (i, j)
$\mathcal{W}_j$	Set of time periods between the early start and late start of activity j, $\mathcal{W}_j = \{ES_j, \cdots, LS_j\}$

Other Parameters:

d_{im} (d_{im})	integer, duration of activity i (j) under mode m
r_{jk}	integer, activity j demand for resource type k
R_k	integer, capacity of resource k
ES_j	integer, earliest start time of activity j
LS_j	integer, latest start time of activity j
p_j	integer, priority of activity j
θ	exponent used to ponderate the priority of activities, $\theta \geq 0$
ε_1	very small duration value assigned to activities with no duration (i.e., milestones), $\varepsilon_1 \geq 0$

ε_2	very small weight used to penalize late scheduling of activities, $\varepsilon_2 \geq 0$
α	parameter used to give tie-breaking benefits to longer duration work (α=1.1) or making each work duration unit equal (α=0), $\alpha \geq 0$
β_{M1}	total alternative shift limit
β_{M2}	alternative mode reduction limit

Decision Variables:

$$x_{jtm} = \begin{cases} 1, & \text{if activity } j \text{ starts at time } t \text{ in mode } m \\ 0, & \text{otherwise.} \end{cases}$$

BIP Model

The obtained binary integer programming (BIP) model is given below.

$$\text{Max} \sum_{j\in\mathcal{J}} \sum_{t\in\mathcal{W}_j} \sum_{m\in\mathcal{M}} \frac{x_{jtm}}{p_j^{\theta}} (\varepsilon_1 + d_{jm})^{\alpha} (1 - \varepsilon_2 t) \tag{2.15}$$

s.t.:

$$\sum_{m\in\mathcal{M}} \sum_{t\in\mathcal{W}_j} t\, x_{jtm} \leq 1 \qquad \forall j \in \mathcal{J} \tag{2.16}$$

$$\sum_{m\in\mathcal{M}} \sum_{t\in\mathcal{W}_j} t\, x_{jtm} \geq \sum_{m\in\mathcal{M}} \sum_{t\in\mathcal{W}_i} (t + d_{im})\, x_{jtm} \qquad \forall j \in \mathcal{J}, \forall (i,j) \in \mathcal{P} \tag{2.17}$$

$$\sum_{m\in\mathcal{M}} \sum_{j\in\mathcal{J}} \sum_{r=max\{t-d_{jm}+1, ES_j\}}^{min\{LS_j,t\}} r_{jk} x_{jbm} \leq R_k \qquad \forall k \in \mathcal{K}, \forall t \in \mathcal{H} \tag{2.18}$$

$$\sum_{j\in\mathcal{J}} \sum_{t\in\mathcal{W}_j} (x_{jt2} + 2x_{jt3} + ... + (M-1)\, x_{jtM}) \leq \beta_{M1} \tag{2.19}$$

$$\sum_{j\in\mathcal{J}} \sum_{t\in\mathcal{W}_j} (x_{jt2} + x_{jt3} + ... + x_{jtM}) \leq \beta_{M2} \tag{2.20}$$

$$x_{jtm} \in \{0,1\} \qquad \forall j \in \mathcal{J}, t \in \mathcal{W}_j \tag{2.21}$$

Equation (2.15) represents the objective function aiming to front-load high-priority activities with long duration. Constraints (2.16) guarantee that each activity is schedule up to once. An activity j in an FS relationship can only be started after its predecessor i has been completed as specified by constraints (2.17). Constraints 2.18

ensure that resources required for a scheduled activity are available when it is executed. Constraints (2.19) govern the amount of time that can be reduced by overtime in an activity. Constraints (2.20) enforce a limit on the total number of activities executed in overtime mode.

Prabhu (2021) developed a minimization variant of the BIP proposed by Bertrand (2020) and used a matheuristic to find approximate solution for moderately large instances of the NSWPP. The objective function used by Prabhu (2021) is given in Equation (2.22).

$$\text{Minimize} \sum_{j\in\mathcal{J}} \sum_{t\in\mathcal{W}_j} \frac{x_{jt}}{p_j^{\theta}} (\varepsilon_1 + d_j)^{\alpha}(t) \tag{2.22}$$

The models developed by Bertrand (2020) and Prabhu (2021) are single calendar and use only the Finish-to-start relationship.

2.3 RCPSP Solution Methods for NSWPPs

The combinatorial nature and intrinsic complexity of RCPSPs have given rise to major contributions during the last four decades. A comprehensive review of early and recent developments in theory building and application of exact, heuristic, and metaheuristic solution methods for RCPSPs can be found according to the different scheduling environments. In Single-project scheduling environment, the RCPSP is applied to schedule a single project with a set of tasks and resources. The goal is to find the optimal schedule that minimizes the completion time of the project or some other objective function (Demeulemeester et al., 1994). In Multi-project scheduling environment, the RCPSP is applied to schedule multiple projects, each with its own set of tasks and resources. The goal is to find the optimal schedules for each project while taking into account the shared resources and inter-project dependencies (Kolisch, 2013). In dynamic scheduling environment, the RCPSP is applied to schedule a project in real-time, taking into account the changing conditions and resource availability. The goal is to find the optimal schedule as the project progresses (Chakrabortty et al., 2020). Under the distributed scheduling environment, the RCPSP is applied to schedule a project across multiple locations or sites. The goal is to find the optimal schedules for each location while taking into account the communication and transportation costs between locations (Stiti & Driss, 2019). For stochastic scheduling environments, the RCPSP is applied to schedule projects in the presence of uncertainty (Herroelen & De Reyck, 1999). The goal is to find the optimal schedule that takes into account the probability of disruptions and other risks. For the majority of well-known RCPSP variants, it is evident that instances with many activities and complex constraints are intractable and cannot solved by commercial solvers. Each of these scheduling environments presents its own challenges and requires a different approach to solving the RCPSP. For example, dynamic scheduling requires real-time optimization techniques, while stochastic scheduling requires probabilistic methods (Couch, 2016).

The focus of Bertrand (2020) , Yin (2022) and Prabhu (2021) has been on the formulation of the NSWPP and the implementation of solution method capable of yielding high-quality solutions for medium and large-scale problem instances within short computation times. This section provides an overview of the exact approaches, heuristics, metaheuristics, constraint programming and satisfiability testing, and matheuristic methods used to solve the RCPSP. Each of these solution approaches has its own advantages and disadvantages, and the choice of method will depend on the specific requirements of the problem, such as the size and complexity of the problem, the time and computational resources available, and the desired level of accuracy. The NSWPP is recently introduced variant of the RCPSP and there are not many efficient solution methods for large-scale instances.

2.3.1 Exact Solution Methods

State-of-the-art exact solvers such as Coin-OR branch and cut (CBC), CPLEX, Gurobi, and Gusek/GLPK contain highly efficient algorithmic implementations of the Simplex method. In particular, in the case of mixed-integer optimization, most solvers use branch-and-bound, branch-and-cut algorithms, and some take advantage of a warm-start. Branch-and-bound, Branch-and-cut algorithms outperform all other exact methods for RCPSP models Demeulemeester & Herroelen (1992). The key advantage of exact methods is that they can find the optimal solution for small to medium size RCPSP problems within a reasonable time frame. With the help of advanced optimization software and computer hardware, exact methods can provide a valuable benchmark solution for these problems. However, the RCPSP problem is strongly NP-hard, making it challenging for exact methods to find optimal solutions for larger problems (Blazewicz et al., 1983). A solution for larger size problems may not be feasible in a reasonable time. Exact methods have been widely used in scheduling and planning problems in various domains, including construction, manufacturing, and transportation. These methods have proven to be effective in finding optimal solutions for problems with complex constraints and requirements. However, their performance can be limited by the size and complexity of the problem, making it necessary to consider approximate methods or heuristics for larger problems.

Exact methods can be used to schedule production processes to meet customer demand and minimize waste. The RCPSP can be used to model the production process and find the optimal schedule that meets the production capacity constraints and resource availability (Neumann et al., 2002). In construction, exact methods can be used to schedule construction activities to minimize the duration of the project and reduce costs. The RCPSP can be used to model the construction process and find the optimal schedule that meets the resource constraints and project requirements (Xie et al., 2021). Exact methods can also be used to schedule supply chain activities, such as transportation and warehousing, to minimize the cost of the supply chain and maximize customer satisfaction. The RCPSP can be used to model the supply chain process and find the optimal schedule that meets the resource constraints and customer demand (Tirkolaee et al., 2020). Exact methods can be used to allocate resources, such as machines and personnel, to various tasks in a project. The RCPSP can be used to model the resource allocation problem and find the optimal allocation that meets the resource constraints and project requirements (Daniels et al., 1996; Jiang & Shi, 2005). These exact methods have proven to be effective in a variety of scheduling and planning problems and continue to be used for the NSWPP.

2.3.2 Constraint Programming

Constraint programming (CP) has been an active area of research since the 1980s and it has gained popularity in recent years due to the increasing complexity of problems that need to be solved and the need for efficient and effective methods for solving them. CP is a mathematical modeling technique that can be used to solve a wide range of combinatorial optimization problems, including the Resource Constrained Project Scheduling Problem (RCPSP). When applying CP to the RCPSP, the problem is first modeled by defining the decision variables, such as the start times and duration of tasks, and the constraints, such as resource availability and precedence relationships between tasks (Hooker, 2002). The constraints can be represented using a variety of formal languages, such as the Constraint Handling Rules (CHR) or the Constraint Logic Programming (CLP) languages (Bistarelli et al., 2004). Once the model is defined, a CP solver is used to find a solution that satisfies all constraints. The CP solver can be based on a variety of search and optimization techniques, such as local search, branch-and-bound, and branch-and-cut. One of the key advantages of constraint programming is its ability to handle complex and non-linear constraints. It allows for the modeling of complex relationships between variables, such as precedence constraints, resource constraints and time windows, and also allows for the representation of uncertain or incomplete information. Additionally, CP can be combined with other optimization methods, such as linear programming, to further improve the quality of the solutions (F. Rossi & Walsh, 2006). CP has been applied to solve many variants of the RCPSP, including multi-mode and multi-project scheduling, and also has been combined with other techniques such as metaheuristics and satisfiability testing for solving more complex instances of the problem (Schnell & Hartl, 2017; H. Li et al., 2018).

Satisfiability testing, also known as the SAT problem, is the problem of determining whether a given Boolean formula is satisfiable, or equivalently, whether there exists an assignment of Boolean values to the variables of the formula that makes it evaluate to true. The SAT solver has a disadvantage in solving large and complex models. It is difficult to represent the RCPSP because of the constraints on the boolean variables and the various constraints (Coelho & Vanhoucke, 2014). However, a combination of Satisfiability testing and Constraint programming has been used to tackle the RCPSP (de Azevedo et al., 2021).

2.3.3 Heuristics and Metaheuristics

Heuristics have been developed to solve the RCPSP, as the problem is known to be NP-hard and finding an exact solution for large-scale problems can be infeasible in a reasonable amount of time. Kelley & E (1963) introduced a priority rules schedule generation scheme. Priority rules are used for sorting activities to be considered for scheduling while avoiding resource conflicts in the RCPSP. Kolisch & Hartmann (1999) introduced a heuristic based on critical path analysis, performing a forward and backward pass on the network of tasks to determine the earliest and latest start times for each task. Another popular heuristic is the Earliest Due Date (EDD) priority rule, which is a simple heuristic that schedules tasks in order of their due dates with the earliest due date task being scheduled first. The EDD rule has been widely used in practice and has been shown to be effective in solving RCPSP problems with a small number of tasks Ballestín et al. (2006). The shortest processing time (SPT) rule is another simple heuristic that schedules tasks in order of their processing time with the shortest processing time task being scheduled first. The SPT rule has been shown to be effective for RCPSP problems with a limited number of resources. Most heuristic methods studied since the 1990's have single-pass or multi-pass priority rules methods (Tormos & Lova, 2003).
(Bertrand, 2020) compared the efficiency performance of certain priority rules proposed by many researchers to identify a proper rule for the naval maintenance facility. Multiple objective functions including Makespan, and Priority 1 Duration-Weighted Centroid (DWC) were tested using different priority sorting schemes. The most suitable heuristic for NSWPP is concluded to be the serial SGS sorted by priority, ES, duration, and priority. Yin (2022) modified the basic serial SGS to deal with multiple calendars, multiple execution modes, precedence relationships (FS, SS, FF, FS) and activity time-enforced constraints.

In the context of RCPSP, metaheuristics have been widely used to find high-quality solutions in a relatively short amount of time. Some popular metaheuristics used to solve the RCPSP include Genetic Algorithms (Alcaraz et al., 2003), Ant Colony Optimization (Merkle et al., 2002), Particle Swarm Optimization (Jarboui et al., 2008), and Simulated Annealing (Józefowska et al., 2001). These methods work by generating and improving candidate solutions through random exploration and exploitation of the solution space. Metaheuristics have been shown to be effective in solving RCPSP problems with large amounts of data, complex constraints, and multiple objectives (Pellerin et al., 2020). The Genetic Algorithm (GA) is a metaheuristic that is based on the principles of natural selection and genetic evolution (Kolisch & Hartmann, 1999). GA has been applied to the RCPSP and has been shown to be effective in finding high-quality solutions in a reasonable amount of time (Hartmann, 1998). Simulated Annealing approach is another metaheuristic inspired by annealing in metallurgy. It uses randomness and a temperature control to search for a near-optimal solution (Cho & Kim, 1997). Tabu search is a local search method that uses a memory of recently visited solutions to avoid getting stuck in a local optimum (Thomas & Salhi, 1998).

2.3.4 Matheuristic Methods

In this section, we will review most important decomposition methods that are closely related to the method proposed in this thesis.

As discussed previously, both heuristics methods and mathematical programming methods have been used to solve the RCPSP. Mathematical programming methods give an optimal solution for scheduling problems. However, the limitation of mathematical programming is that it is difficult to find an optimal solution for large size problems. Although heuristic methods only find feasible solutions, they can handle large and complex scheduling problems. A large complex optimization problem can be solved by hybrid solution methods, which combine both (meta)heuristics and mathematical programming methods. A hybrid solution method is called a matheuristic. These matheuristics are increasingly being used to solve scheduling problems in industry.

Matheuristics can be classified into three approaches:

1. Decomposition approaches. Decomposition approaches are a class of optimization methods that involve breaking down a large and complex problem into smaller and simpler subproblems that can be solved independently (Prabhu, 2021).

2. Improvement heuristics or metaheuristic. They work by exploring the solution space and making small improvements to the current solution in order to find better ones. They are particularly very useful in solving problems that are difficult or impossible to solve exactly such as RCPSP, which is a NP-Hard problem (Maniezzo et al., 2021).

3. Relaxation-based approaches. These solution approaches find an approximate solution by relaxing certain or all constraints in the original problem. The solution of the relaxed problem is then used as a starting point for finding a solution to the original problem (An et al., 2021).

Several authors have addressed the rescheduling problem for single-stage, single-unit production processes using a decomposition method (Roslöf et al., 2001). Their proposed solution includes a MILP formulation for inserting batches of activities into a schedule. A set of (direct and indirect) precedence constraints is used to maintain the relative order of the activities in the batches. A similar approach can be used for scheduling single-stage production process with parallel units as in Méndez & Cerdá (2003) where a decomposition approach is used to generate an initial schedule. Roslöf et al. (2002) used a MILP formulation to add new batches of activities to the current partial schedule while maintaining the relative order of the scheduled batches of a small-scale industrial problem.

Since the development of hybrid methods such as model-reduction methods, aggregation techniques, and decomposition methods (Chen, 2002; Yee, 1998; Harjunkoski & Grossmann, 2002), the use of exact methods to solve large-scale scheduling problems with complex constraints has become increasingly attractive (Maravelias, 2006; Roslöf et al., 2001). These methods are mainly based on exploiting the flexibility provided by mixed-integer linear programming (MILP) to easily accommodate complex technological constraints while restricting the number of simultaneous decisions in order to ensure reasonable CPU time.

A process-specific MILP is tailored to the constraints and requirements of a specific process, such as scheduling, routing, or resource allocation. The use of a process-specific MILP model allows to take into account all the details of the problem and find optimal solutions. The model is reduced in size by using pre-processing rules (Méndez & Cerdá, 2002). In the same way, applying the block-planning technique, by which batches within a block are predetermined based on sequence-dependent changeover times can also lead to significant improvement in performance of RCPSP solution (Günther & Neuhaus, 2006).

In batch processing, a set of similar or identical tasks are grouped together and processed as a batch with the goal of improving efficiency and reducing costs. Batch processing is commonly used in industries such as chemical, pharmaceutical, and food processing, as well as in manufacturing (Trautmann & Schwindt, 2005). Priority rules are used in batch processing to determine the order in which batches are processed. These rules are usually based on factors such as the urgency of the task, the availability of resources, and the cost of processing the batch. Common priority rules include First-Come-First-Served (FCFS), Earliest Due Date (EDD), Shortest Processing Time (SPT), and Critical Ratio (CR) (Trautmann & Schwindt, 2005). The relative order of batches can be important in batch processing as it can affect the overall efficiency and effectiveness of the process. For example, in the pharmaceutical industry, batches must be processed in a specific order to ensure that the final product meets regulatory requirements (Schwindt & Trautmann, 2000). To maintain the relative order of batches in batch processing, various techniques have been proposed, such as using priority rules and constraints within a mathematical model, such as a mixed-integer linear programming (MILP) model. In some cases, pre-processing procedures can be used to eliminate redundant variables and constraints within the MILP model, resulting in improved computational efficiency. However, there are limitations to these techniques, and sometimes the relative order may not be maintained

(Maravelias, 2006).

A batch-based MILP model is a mathematical model used to solve RCPSPs. In this model, the activities of a project are divided into a series of smaller, manageable batches, and the resources required for each batch are optimized over time. The objective is to minimize the makespan or the total completion time of the project, subject to constraints such as resource availability, task precedence, and time windows. The advantage of using a batch-based MILP model is that it provides a more fine-grained control over the scheduling process, allowing for more accurate and efficient scheduling decisions (Afshar-Nadjafi, 2021).

A network-based MILP model is a type of mathematical optimization model that represents the RCPSP as a network of activities and resources. In this model, the activities are represented as nodes in the network, and the precedence constraints between the activities are represented as directed edges. The resources are modeled as constraints on the use of each activity. The objective of the model is to determine the optimal schedule for the activities such that the project completion time is minimized while satisfying all the resource constraints. This type of model is formulated as a mixed-integer linear programming (MILP) problem and can be solved using optimization algorithms such as branch-and-bound or branch-and-cut (Alipouri et al., 2020).

A batch-based MILP model and a network-based MILP model are proposed for scheduling individual groups. Kopanos et al. (2010) propose a solution method consisting of two steps: a constructive step and an improvement step. Two alternatives batch-based MILP formulations are proposed for scheduling the activity batches in the constructive step based on the number of suitable processing units. Reordering and reinsertion are part of the improvement step. The reordering stage involves swapping the order of activity batches processed to improve the workflow. Reinsertion consists of the rescheduling of individual activity batches. The two stages are repeated until no improvement is achieved within a predetermined number of iterations. The cumulative model tends to have a higher computational cost. The best results were obtained when a single batch was scheduled per iteration in an experimental study examining the effect of group size (Kopanos et al., 2010). A similar decomposition method is applied to a real-world application of a multi-stage, single-unit problem (Kopanos et al., 2012). According to Gomes et al. (2010), a discrete-time network-based model can be used to insert additional activity batches into an existing schedule for a production process with identical parallel processing units.

In the decomposition method, the short-term scheduling problem is divided into two subproblems: the master problem and the subproblem. The master problem is a linear programming problem that defines the overall objective function and the constraints that are common to all subproblems. The subproblem is a mixed integer linear programming problem that defines the objective function and the constraints that are specific to each subproblem. The master problem and the subproblem are solved iteratively, with the solution of one problem being used as input for the other, until a satisfactory solution is found (Trautmann et al., 2008).

The short-term scheduling problem has been widely discussed in the literature. This decomposition approach is based on a mixed-inter linear programming formulation broken down into a set of shorter-term scheduling interdependent sub-problems. Batching provides a set of batches with fixed sizes needed to satisfy the primary requirements of the problem. Batch scheduling allocates the activities, duration, and mode over time to the processing of activities in each batch.

Prabhu (2021) proposed a multi-step optimization matheuristic based on solving MILP formulations repetitively for carefully grouped subsets of activities using three ranking criteria (priority, earliest start, and duration). Their proposed multi-step optimization (MSO) has three main steps: sorting activities, creating subproblems consisting of activities in subgroups, and repetitively optimizing the schedule until all subgroups are processes. Prabhu (2021) developed three variants of the MSO approach as illustrated in Figure 2.1.

The first variant (MSO-1) mainly divides the list of activities into two subgroups: one with priority-1 activities, and the other with priority 2 and 3 activities. The

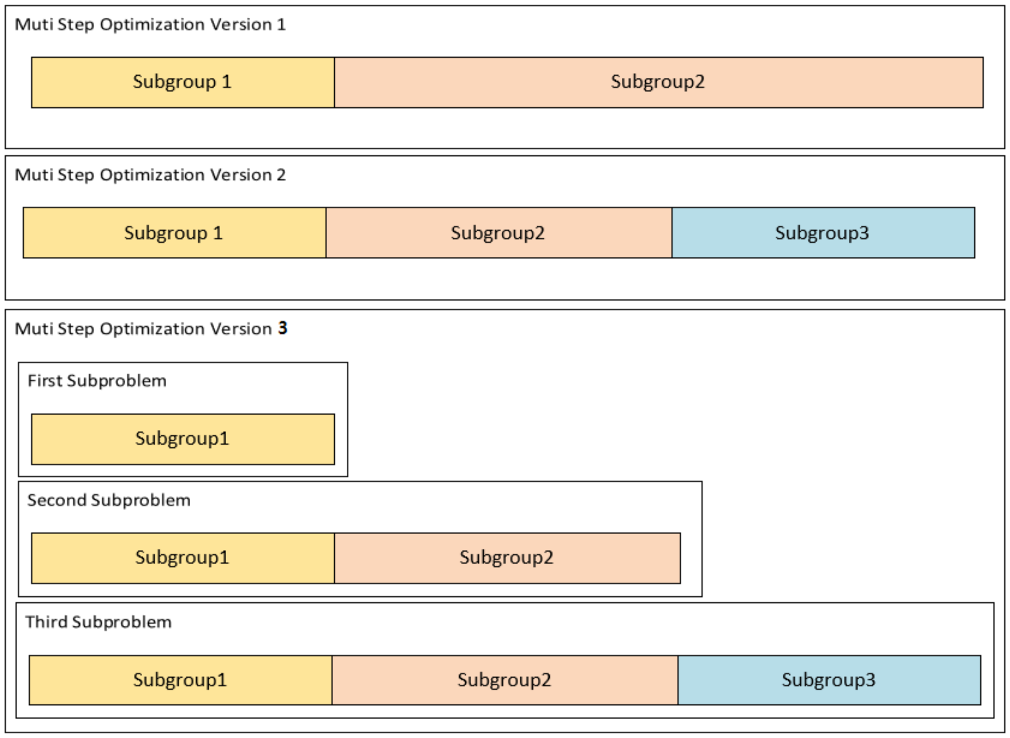

Figure 2.1: Three variants of the MSO algorithm (Prabhu, 2021)

optimization model is then run on the first subgroup with priority 1 activities. The solution obtained is then used to "fix or set" the ES and LS times of the activities in the subgroup. Then, the optimization model is run on the activities in the second subgroup.

MSO-2 the second variant creates subgroups of a predetermined size. First , the network activities are first sorted by decreasing priority, increasing ES, and decreasing duration. Once the activities have been sorted, subgroups of the appropriate size are created. The algorithm then proceeds to optimizing the first subgroup. The obtained solution is used to "fix or set" the start times of the activities in the subgroup. The algorithm then proceeds to the next iteration where the next subgroup is optimized and the start times of its activities determined and fixed. This process is repeated until all subgroups have been optimized.

The third variant MSO-3 was developed with the aim of taking into account the precedence relationships. Hence, the subgroups are not optimized independently as in MSO-2. Starting with the first subgroup, the algorithm first optimizes the current batch then uses its solution to “fix or set” the start times of the activities in the batch. Then, the algorithm adds the next subgroup to the current batch with information about the precedence pairs, duration, availability of resources and demand for resources to create a new batch of activities to optimize. This process repeats until the final batch that contains all network activities is optimized. Numerical experiments conducted by Prabhu (2021) show that MSO 3 is the most time efficient variant and produces higher quality solutions. MSO-3 provides better quality solutions compared to the serial-SGS, however the serial-SGS is capable if producing a solution for large size problems in significantly shorter times.

2.4 Network Complexity Measures

There are four general categories of network complexity measures: precedence-oriented, resource-oriented, time-oriented and hybrid coefficients.

Coefficient of Network complexity (CNC).

The precedence-oriented CNC measures how dense the project network is. It is the ratio of the total number of non-dummy precedence pairs (P) to the total number of non-dummy activities (n).

$$CNC = \frac{P}{n} \tag{2.23}$$

If many precedence pairs are present, a project is considered to be a disjunctive project. In a disjunctive project many tasks cannot be carried out concurrently. On the other hand, cumulative projects are those with a low number of precedence pairs. In a cumulative project many activities can be carried out simultaneously. A number of studies have indicated that problems become easier as their CNC increases (Herroelen & De Reyck, 1999).

Order Strength (OS).

The Order Strength (OS) is another precedence-oriented coefficient aiming to indicate network density. The total number of precedence relationships is denoted by $\hat{P}$ including both direct and transitive precedence relationships. $\frac{n^2-n}{2}$ is the theoretical maximum number of all possible precedence relationships.

$$OS = \frac{\hat{P}}{\frac{n^2 - n}{2}} \tag{2.24}$$

The OS is similar to the CNC as it also measures the density of the network (Herroelen & De Reyck, 1999; De Nijs, 2013). Small values of the CNC and OS indicate that the project has many activities that can be scheduled in parallel Koné et al. (2011). Smaller values of CNC and OS result in longer solution time to find an optimal solution resulting from a wider feasible region (Herroelen & De Reyck, 1999).

Resource Factor (RF).

(RF) is a resource-oriented complexity coefficient which indicates the average resource demand per activity for a resource type.

$$RF = \frac{1}{n\,|\mathcal{K}|} \sum_{i \in \mathcal{J}} \sum_{k \in \mathcal{K}} \begin{cases} 1, & \text{if } r_{ik} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.25}$$

where n is the number of activities, $\mathcal{J}$ is the set of activities, $\mathcal{K}$ is the set of resource types, and r_{ik} is the demand for resource type k by activity i.

The RF measures the density of the resource demand based on the demand matrix r_{ik}. It scans for each activity/resource combination whether the resource is requested by the activity or not and calculates the average fraction of all resources requested by all activities (percentage of resource use). If $RF = 1$, then all resources are demanded by all activities. If $RF = 0$, then none of the resources are demanded by any of the activities. Kolisch et al. (1995) conducted experiments on 480 RCPSP instances with 30 activities and four types of resources to demonstrate that CPU time increases with higher RF.

Resource Constrainedness (RC_k).

Resource Constrainedness (RC_k), a resource-oriented network complexity coefficient, defines the average usage of resource type k over the activities using that resource k.

$$RC_k = \frac{\sum\limits_{i \in \mathcal{J}} r_{ik}}{R_k \sum\limits_{i \in \mathcal{J}} \begin{cases} 1, & \text{if } r_{ik} > 0 \\ 0, & \text{otherwise} \end{cases}} \tag{2.26}$$

where R_k is the availability for resource type k.
RC indicates the average amount of resources Constrainedness over all resources (De Nijs, 2013; Kolisch et al., 1995; Patterson, 1976). It is calculated by:

$$RC = \frac{\sum\limits_{k \in \mathcal{K}} RC_k}{|\mathcal{K}|} \tag{2.27}$$

A large value of RC indicates an extremely resource-constrained project. As demonstrated in Figure 2.2, computation time increases as the RC increase at first, then drops after it reaches a peak (Herroelen & De Reyck, 1999). It is interesting to note that the RC exhibits an easy-hard-easy complexity pattern similar to a bell curve. It is not very difficult to solve problem instances in which RC is extremely small or extremely large. A higher computational effort is required to solve instances with RC values between 0.4 and 0.75.

Resource Strength (RS)
Resource Strength (RS) is a measure of the resources available for performing tasks or activities. It is a measure of how much resource (e.g., labor, material, or equipment) is available to be used in a particular activity or set of activities.

$$RS_k = \frac{R_k}{\frac{1}{n} \sum\limits_{i \in \mathcal{J}} r_{ik}} \tag{2.28}$$

RS is often used in scheduling and project management to evaluate the feasibility of a project schedule, as well as to identify potential resource constraints and bottlenecks. The higher the RS, the more resources are available to be used for completing tasks, which can result in a faster solution or more efficient schedule. Conversely, a low RS can indicate that there may not be enough resources to complete all the

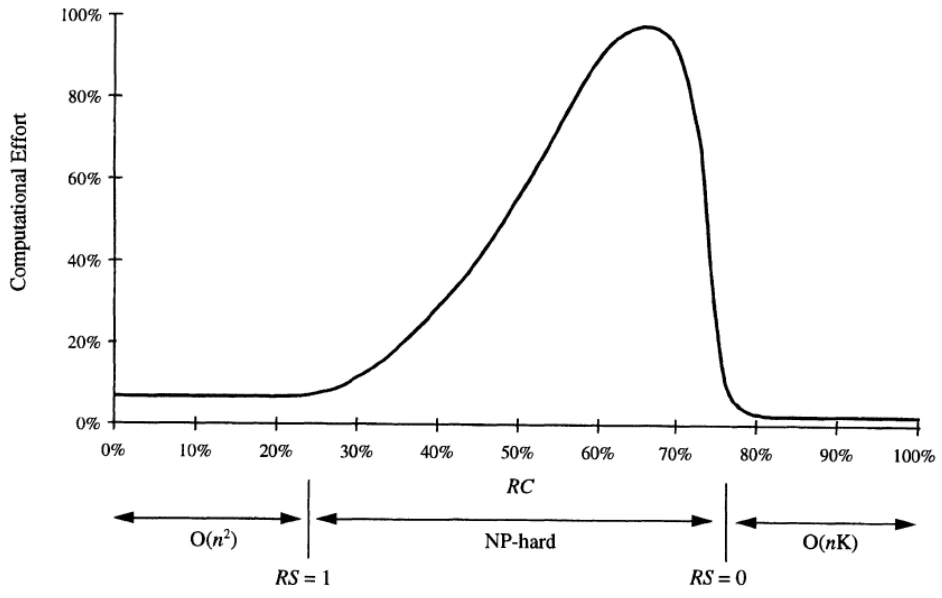

Figure 2 Computational complexity *vs* RC and RS.

Figure 2.2: Computational complexity vs RC (Herroelen & De Reyck, 1999)

tasks in a schedule, or that the schedule may take longer or be more complex due to resource constraints (Cooper, 1976).

Process Range (PR).

The quotient between the maximum and minimum duration of project activities is known as the Process Range (PR) as shown in Equation (2.29). If the PR is large, this means there are large duration differences between activities, which in turn increases the computation time because the solver must search through a larger solution space to assign an optimal start time for the longer-duration activities that may exist among the short-duration activities (Herroelen & De Reyck, 1999).

$$PR = \frac{\max\limits_{i=1 \dots n} \{D_i\}}{\min\limits_{i=1 \dots n} \{D_i\}} \tag{2.29}$$

Disjunction Ratio (DR).

Disjunction Ratio (DR) is the product of the coefficient of network complexity (CNC) with the Resource Constrainedness (RC).

$$DR = CNC \times RC \tag{2.30}$$

Koné et al. (2011) proposes an MILP formulation type selection guide based on the DR and PR indicators values as illustrated in Table 2.3 where the symbol $\succ$ means "dominate".

Table 2.3: Performance of MILP formulations based on DR and PR (Yin, 2022)

	High DR	**Low DR**
Low PR	DDT $\succ$ DT$\succ$ FCT $\succ$ OOE_x $\succ$ SEE	DDT $\succ$ DT $\succ$ OOE_x $\succ$ FCT $\succ$ SEE
High PR	FCT,OOE_x $\succ$ SEE $\succ$ DT $\succ$ DDT	OOE_x $\succ$ FCT $\succ$ SEE $\succ$ DT $\succ$ DDT

In this thesis, the standard NSWPP formulation is extended to the multi-calendar cases with additional precedence relationships and time constraints. The following chapters provide detailed problem definition, mathematical modeling and solution approaches followed by extensive numerical experiments and their discussion.

Chapter 3

Mathematical Model and Solution Approaches

The Naval Surface Ship Work Period Problem (NSWPP) is a complex scheduling problem that involves coordinating multiple maintenance and refit activities on a naval surface ship. The problem is characterized by a large number of activities that are governed by multiple calendars for activities and resources. Additionally, there are often other than Finish-to-Start precedence relationships that must be taken into account, as well as timing constraints that dictate when an activity can start or end. Another level complication is added when activities are permitted to be executed in different modes requiring different amount of resources and resulting in different activity duration. The objective is to front-load high-priority and long-duration activities during the project time horizon.

Initially, the original NSWPP (without the multi-calendar, multi-mode activities and time enforcement constraints) was formulated as an extension of the traditional RCPSP problem. (Bertrand, 2020) and (Prabhu, 2021) have attempted to apply and compare the performance of most RCPSP optimization modeling and solution techniques in the literature to solve the NSWPP. Differences in approach performance was observed between the two optimization problems although they both share several elements.

The problem considered in this thesis aims at including the new requirements and characteristics identified by the industrial partners of this research project. The requirements and characteristics include multi-calendar activities, multi-calendar resources, three additional activity relationships (FF, SS, and SF), and activity start and/or finish time constraints. Adapting each of these new requirements add a level of challenge to solve the problem.

This chapter is organized as follows. The first section will present the universal matrix developed to accommodate the multi-calendar requirements. Then the developed BIP formulation will be presented.

3.1 Universal Duration Matrix

To deal with multiple calendars, Yin (2022) proposed a universal duration matrix that converts all calendars into a single one. Once the activity durations are set in the universal calendar, the mathematical formulation can be developed as if we were dealing with one calendar.

The following example is presented here to illustrate the process of building the duration matrix as developed by Yin (2022). Table 3.1 shows the duration of two activities following two different calendars. Activity 1 has a duration of 3 days and follows a 7-day calendar while activity 2 has a duration of 5 days and follows a 5-day calendar (i.e., no work on weekends). It also is assumed that day 5 (Friday) of the week under consideration is a holiday and no work can take place in both calendars.

Table 3.1: Illustrative example with two activities

Activity	Duration	Calendar
1	3	7-day
2	5	5-day

Table 3.2 shows a calendar representation of the basic duration matrix $\boldsymbol{d}$ obtained for the example considered. Element d_{jt} represents the effective duration of activity j if it were to start on day t. Q is a large positive number. Days when work is prohibited or not allowed because of calendar requirements have duration values of $-Q$ to prevent their selection by the optimization model. The value of Q must be selected to avoid scaling issues. A decent value would be slightly larger than the planning horizon to be determined later with the serial SGS heuristic.

Table 3.2: Calendar representation of the duration matrix $\boldsymbol{d}$

Days	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed
t	1	2	3	4	5	6	7	8	9	10
1	3	3	4	4	$-Q$	3	3	3	3	3
2	8	8	8	8	$-Q$	$-Q$	$-Q$	5	7	7

$$\boldsymbol{d} = \begin{pmatrix} 3 & 3 & 4 & 4 & -Q & 3 & 3 & 3 & 3 \\ 8 & 8 & 8 & 8 & -Q & -Q & -Q & 5 & 7 \end{pmatrix}$$

For example, $d_{11} = 3$ means that the effective duration of activity 1 is 3 if it is started on day 1. Indeed, if activity 1 starts at the beginning of day 1 (Monday) it will finish at the end of day 3 (Wednesday), hence its duration is 3 full days. $d_{13} = 4$ means that the effective duration of activity 1 is 4 if it is started on day 3. The duration is extended by one day because the fourth day of the workweek would be Friday which is a holiday. Thus, the third day of actual work is Saturday or day 6. So that the effective duration in the universal calendar is 4 days. Similarly, we get $d_{21} = 8$ meaning that the effective duration of activity 2 is 8 if it is started on day 1 because its duration spans the holiday and the weekend, thus adding three days to the effective duration in the universal calendar.

As exposed earlier, under the NSWPP naval vessels tend to have limited time to undergo refit activities. NSWPPs also have timing constraints on activities that prescribe their start dates and/or end dates. With these timing constraints and complex precedence relationships, it is highly plausible for a problem to be infeasible if some activities are not crashed (*i.e.,* duration reduced). Multi-mode execution of activities allow the modelling of varying activity duration. These processing modes include overtime and other execution modes with decreasing activity duration in the multi-mode model. To include the multi-mode characteristic in the universal calendar extended problem formulation, the duration matrix is extended from two dimensions to three dimensions where each element of the matrix is a row vector with M elements.

M is the number of execution modes available. Mode 1 is the normal duration mode while mode M is the smallest duration mode. The m^{th} element d_{jtm} of the row vector at the intersection of row j and column t of matrix $\boldsymbol{d}$ is the effective duration of activity j under execution mode m if it were to start on day t. In general, we have:

$$\boldsymbol{d} = \begin{pmatrix} [d_{111}, \cdots, d_{11M}] & [d_{121}, \cdots, d_{12M}] & \cdots & [d_{1t1}, \cdots, d_{1tM}] & \cdots & [d_{1H1}, \cdots, d_{1HM}] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \boldsymbol{[\cdots, d_{jtm}, \cdots]} & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ [d_{n11}, \cdots, d_{n1M}] & \cdots & \cdots & [d_{nt1}, \cdots, d_{ntM}] & \cdots & [d_{nH1}, \cdots, d_{nHM}] \end{pmatrix}$$

For the illustrative example introduced at the start of this section in Table 3.1, assuming that activity 1 can only be reduced by one day and activity 2 can be reduced by at most two days, the following duration per execution modes are defined as depicted in Table 3.3.

Table 3.3: Activity duration per execution mode

	Modes		
Activity	1	2	3
1	3	2	2
2	5	4	3

The following transfer matrix is then obtained:

$$\boldsymbol{d} = \begin{pmatrix} [3,2,2] & [3,2,2] & [4,3,3] & [4,3,3] & [-Q,-Q,-Q] & \cdots \\ [8,7,6] & [8,7,6] & [8,7,6] & \boldsymbol{[8,7,6]} & [-Q,-Q,-Q] & \cdots \end{pmatrix}$$

Vector $[8, 7, 6]$ at the intersection of row 2 and column 4 gives the three effective duration values for activity 2 if it where to start on day 4. Under mode 1 the duration is 8, under mode 2 the duration is 7 (reduced by one day), and under mode 3 the duration is 6 (reduced by one additional day).

3.2 Earliest Start and Latest Start

This subsection recalls the formulas developed by Yin (2022) to determine the $[ES, LS]$ windows for multi-mode problems. Table 3.4 shows that the $[ES, LS]$ windows are the same for Start On, Start On or Before, and Start On or After, but different for the other three. For multi-mode problems, the start on/before/after constraints can still be satisfied by pre-calculating $[ES, LS]$ because the duration of the activity is not needed. On the other hand, the windows must be determined for Finish on, Finish on or before, and Finish on or after.

Table 3.4: ES and LS values for activity time constraints with multiple execution modes (Yin, 2022)

Start On	ES	$date$
	LS	$date$
Start On or Before	ES	$\max\{0, ES_{FS}, ES_{FF}, ES_{SS}, ES_{SF}\}$
	LS	$\min\{date, LS_{FS}, LS_{FF}, LS_{SS}, LS_{SF}\}$
Start On or After	ES	$\max\{date, ES_{FS}, ES_{FF}, ES_{SS}, ES_{SF}\}$
	LS	$\min\{H, LS_{FS}, LS_{FF}, LS_{SS}, LS_{SF}\}$
Finish On	ES	$date - db_{j,date,1}$
	LS	$date - db_{j,date,M}$
Finish On or Before	ES	$\max\{0, ES_{FS}, ES_{FF}, ES_{SS}, ES_{SF}\}$
	LS	$\min\{date - db_{j,date,M}, LS_{FS}, LS_{FF}, LS_{SS}, LS_{SF}\}$
Finish On or After	ES	$\max\{date - db_{j,date,1}, ES_{FS}, ES_{FF}, ES_{SS}, ES_{SF}\}$
	LS	$\min\{H, LS_{FS}, LS_{FF}, LS_{SS}, LS_{SF}\}$

3.3 BIP Model

A Binary integer programming (BIP) model based on the one introduced in Yin (2022) is developed to find the optimal solution for this specific problem described above. The objective function however is changed from maximization to minimization because it was empirically observed that minimization models of the NSWPP could be solved faster than their maximization counterparts (Prabhu, 2021). Minimization problems have a well-established duality theory, which can be used to obtain additional information about the solution space and to generate solutions more efficiently. In some scheduling problems, it may be more natural to express the objective as a minimization problem. For example, minimizing makespan instead of maximizing throughput. More in depth discussion of advantage and disadvantage of each modeling approach will included in Chapter 4 dealing with the numerical experiments.

3.3.1 Notation

The following notation is used to formulate the scheduling optimization model.

Indices:

i	index of activities
j	index of activities
t	index of time periods (e.g., days)
m	index of execution modes

Parameters for Sets:

n	integer, number of non-dummy activities
H	integer, time horizon considered for scheduling
K	integer, number of resource types
M	integer, number of execution modes

Sets:

$\mathcal{J}$	Set of activities, $\mathcal{J} = \{0, \cdots, n\}$ with index i or j
$\mathcal{H}$	Set of time periods, $\mathcal{H} = \{0, \cdots, H\}$ with index t
$\mathcal{K}$	Set of types of resources, $\mathcal{K} = \{1, \cdots, K\}$ with index k
$\mathcal{M}$	Set of execution modes, $\mathcal{M} = \{0, \cdots, M\}$ with index m
$\mathcal{P}_{FS}$	Set of immediate finish to start activity pairs (i, j)
$\mathcal{P}_{SF}$	Set of immediate start to finish activity pairs (i, j)
$\mathcal{P}_{FF}$	Set of immediate finish to finish activity pairs (i, j)
$\mathcal{P}_{SS}$	Set of immediate start to start activity pairs (i, j)
$\mathcal{W}_j$	Set of time periods between the early start and late start of activity j, $\mathcal{W}_j = \{ES_j, \cdots, LS_j\}$
$\mathcal{F}_o$	Set of activity and time pairs (i, s) specifying the time s when activity i must Finish On
$\mathcal{F}_b$	Set of activity and time pairs (i, s) specifying the time s when activity i must Finish On or Before

$\mathcal{F}_a$	Set of activity and time pairs (i, s) specifying the time s when activity i must Finish On or After
$\mathcal{S}_o$	Set of activity and time pairs (i, s) specifying the time s when activity i must Start On
$\mathcal{S}_b$	Set of activity and time pairs (i, s) specifying the time s when activity i must Start On or Before
$\mathcal{S}_a$	Set of activity and time pairs (i, s) specifying the time s when activity i must Start On or After

Other Parameters:

d_{im} (d_{jm})	integer, baseline duration of activity i under mode m
d_{itm} (d_{jtm})	integer, duration of activity i under mode m if starting at time t based on its respective calendar
r_{jk}	integer, activity j demand for resource type k
R_k	integer, capacity of resource k
ES_j	integer, earliest start time of activity j
LS_j	integer, latest start time of activity j
p_j	integer, priority of activity j
θ	exponent used to ponderate the priority of activities, $\theta \geq 0$
ε_1	very small duration value assigned to activities with no duration (i.e., milestones), $\varepsilon_1 \geq 0$
β	parameter used to control the weight of execution mode in the objective function, $\beta \geq 0$
α	parameter used to control the weight given to activity duration in the objective function, $\alpha \geq 0$

Decision Variables:

The formulation contains a binary variable for activities determining the start time of the activities:

$$x_{jtm} = \begin{cases} 1, & \text{if activity } j \text{ starts at time } t \text{ in mode } m \\ 0, & \text{otherwise.} \end{cases}$$

3.3.2 Objective Function

Due to the limited time available to conduct refit operations, it is not always possible to perform all maintenance activities before the naval vessel has to start its next mission. Therefore, the objective of the problem in Equation (3.1) is to front-loading high priority and long-duration activities as much as possible. To avoid artificially prioritizing activities that see their duration increased due to holidays or days off in their respective calendar, the baseline activity duration (d_{jm}) is used in the objective function. The execution mode (M) is assumed to be the normal duration mode for all activities.

$$\text{Minimize} \sum_{j \in \mathcal{J}} \sum_{t \in \mathcal{W}_j} \sum_{m \in \mathcal{M}} \frac{x_{jtm}}{p_j^{\theta}} (\varepsilon_1 + d_{jm})^{\alpha}\, t\, m^{\beta} \tag{3.1}$$

3.3.3 Constraints

1. Each activity must be executed only once within its ES and LS time window.

$$\sum_{t \in \mathcal{W}_j} \sum_{m \in \mathcal{M}} x_{jtm} \leq 1 \qquad \forall j \in J \tag{3.2}$$

2. Activity Precedence relationships.

$$\sum_{t \in \mathcal{W}_i} \sum_{m \in \mathcal{M}} (t + d_{itm}) x_{itm} \leq \sum_{t \in \mathcal{W}_j} \sum_{m \in \mathcal{M}} t x_{jtm} \qquad \forall i \in \mathcal{J}, \forall (i,j) \in \mathcal{P}_{FS} \tag{3.3}$$

$$\sum_{t \in \mathcal{W}_i} \sum_{m \in \mathcal{M}} (t + d_{itm}) x_{itm} \leq \sum_{t \in \mathcal{W}_j} \sum_{m \in \mathcal{M}} (t + d_{itm}) x_{itm} \; \forall i \in \mathcal{J}, \forall (i,j) \in \mathcal{P}_{FF} \tag{3.4}$$

$$\sum_{t \in \mathcal{W}_i} \sum_{m \in \mathcal{M}} t x_{itm} \leq \sum_{t \in \mathcal{W}_j} \sum_{m \in \mathcal{M}} t x_{itm} \qquad \forall i \in \mathcal{J}, \forall (i,j) \in \mathcal{P}_{SS} \tag{3.5}$$

$$\sum_{t \in \mathcal{W}_i} \sum_{m \in \mathcal{M}} t x_{itm} \leq \sum_{t \in \mathcal{W}_j} \sum_{m \in \mathcal{M}} (t + d_{itm} - 1) x_{itm} \qquad \forall i \in \mathcal{J}, \forall (i,j) \in \mathcal{P}_{SF} \tag{3.6}$$

Constraints (3.3)–(3.6) enforce the precedence relationship for all four types of requirements (FS, FF, SS, SF) respectively. Constraint (3.3), for a Finish-to-Start relationship, ensures that a successor j can begin only after its predecessor i has finished. Constraint (3.4) deals with the finish-to-finish relationships by ensuring that the successor j cannot be completed until its predecessor i is finished. Constraint (3.5) deals with the Start-to-Start relationship by ensuring that the successor j can only start if the predecessor i has started. Constraint (3.6) deals with the Start-to-Finish relationship by ensuring that the successor j can finish iff its predecessor i has started. The (-1) term is in the right-hand side of constraint (3.6) because an activity starts at the beginning of a period but finishes at the end of a period. So for an activity to be completed by the beginning of a period, it must have been completed by the end of the previous period (Pritsker et al., 1969).

3. Timing constraints.

   Some activities must start or end on, before or after a certain date due to various operational, logistics or administrative requirements.

$$\sum_{t \in \mathcal{W}_i} \sum_{m \in \mathcal{M}} t x_{itm} = s \quad \forall (i,s) \in \mathcal{S}_o \tag{3.7}$$

$$\sum_{t \in \mathcal{W}_i} \sum_{m \in \mathcal{M}} t x_{itm} \leq s \quad \forall (i,s) \in \mathcal{S}_b \tag{3.8}$$

$$\sum_{t \in \mathcal{W}_i} \sum_{m \in \mathcal{M}} t x_{itm} \geq s \quad \forall (i,s) \in \mathcal{S}_a \tag{3.9}$$

   Constraint (3.7) forces the activity to start exactly at time s. Constraints (3.8) and (3.9) force an activity j to start before or after time s respectively.

$$\sum_{t \in \mathcal{W}_i} \sum_{m \in \mathcal{M}} (t + d_{itm} - 1) x_{itm} = s \qquad \forall (i,s) \in \mathcal{F}_o \tag{3.10}$$

$$\sum_{t \in \mathcal{W}_i} \sum_{m \in \mathcal{M}} (t + d_{itm} - 1) x_{itm} \leq s \qquad \forall (i,s) \in \mathcal{F}_b \tag{3.11}$$

$$\sum_{t \in \mathcal{W}_i} \sum_{m \in \mathcal{M}} (t + d_{itm} - 1) x_{itm} \geq s \qquad \forall (i,s) \in \mathcal{F}_a \tag{3.12}$$

   Constraint (3.10) forces the activity to end exactly at time s. Constraints (3.11) and (3.12) force an activity j to end before or after time s respectively.

4. Resource Constraints.

   Constraints (3.13) guarantee that, in any given period t, the total amount of resource k used by all scheduled activities does not exceed the resource capacity during that period R_{tk}.

$$\sum_{i\in\mathcal{J}} \sum_{s=max\{t-db_{itm}+1,ES_i\}}^{min\{LS_i,t\}} \sum_{m\in\mathcal{M}} r_{jkt}x_{ism} \leq R_{tk} \qquad \forall k\in\mathcal{K}, \forall t\in\mathcal{H} \tag{3.13}$$

5. Bounds constraints.

$$x_{jtm}\in\{0,1\} \qquad \forall j\in\mathcal{J}, t\in\mathcal{W}_j \tag{3.14}$$

Putting the model together gives the following BIP.

$$\text{Min} \sum_{j\in\mathcal{J}} \sum_{t\in\mathcal{W}_i} \sum_{m\in\mathcal{M}} \frac{x_{itm}}{p_j^{\theta}}(\varepsilon_1+d_{im})^{\alpha}\, t\, m^{\beta}$$

s.t.:

$$\begin{aligned}
&\sum_{t\in\mathcal{W}_j}\sum_{m\in\mathcal{M}} x_{jtm} \leq 1 && \forall j\in\mathcal{J}\\
&\sum_{j\in\mathcal{J}} \sum_{b=max\{t-db_{jtm}+1,ES_j\}}^{min\{LS_j,t\}} \sum_{m\in\mathcal{M}} r_{jkt}x_{jbm} \leq R_{tk} && \forall k\in\mathcal{K}, \forall t\in\mathcal{H}\\
&\sum_{t\in\mathcal{W}_i}\sum_{m\in\mathcal{M}} (t+d_{itm})x_{itm} \leq \sum_{t\in\mathcal{W}_j}\sum_{m\in\mathcal{M}} tx_{jtm} && \forall (i,j)\in\mathcal{P}_{FS}\\
&\sum_{t\in\mathcal{W}_i}\sum_{m\in\mathcal{M}} (t+d_{itm})x_{itm} \leq \sum_{t\in\mathcal{W}_j}\sum_{m\in\mathcal{M}} (t+d_{jtm})x_{jtm} && \forall (i,j)\in\mathcal{P}_{FF}\\
&\sum_{t\in\mathcal{W}_i}\sum_{m\in\mathcal{M}} tx_{itm} \leq \sum_{t\in\mathcal{W}_j}\sum_{m\in\mathcal{M}} tx_{jtm} && \forall (i,j)\in\mathcal{P}_{SS}\\
&\sum_{t\in\mathcal{W}_i}\sum_{m\in\mathcal{M}} tx_{itm} \leq \sum_{t\in\mathcal{W}_j}\sum_{m\in\mathcal{M}} (t+d_{jtm}-1)x_{jtm} && \forall (i,j)\in\mathcal{P}_{SF}\\
&\sum_{t\in\mathcal{W}_i}\sum_{m\in\mathcal{M}} t\,x_{itm} = s && \forall (i,s)\in\mathcal{S}_o\\
&\sum_{t\in\mathcal{W}_i}\sum_{m\in\mathcal{M}} t\,x_{itm} \leq s && \forall (i,s)\in\mathcal{S}_b\\
&\sum_{t\in\mathcal{W}_i}\sum_{m\in\mathcal{M}} t\,x_{itm} \geq s && \forall (i,s)\in\mathcal{S}_a\\
&\sum_{t\in\mathcal{W}_i}\sum_{m\in\mathcal{M}} (t+d_{itm}-1)x_{itm} = s && \forall (i,s)\in\mathcal{F}_o
\end{aligned}$$

$$\sum_{t \in \mathcal{W}_i} \sum_{m \in \mathcal{M}} (t + d_{itm} - 1)x_{itm} \leq s \qquad \forall (i,s) \in \mathcal{F}_b$$

$$\sum_{t \in \mathcal{W}_i} \sum_{m \in \mathcal{M}} (t + d_{itm} - 1)x_{itm} \geq s \qquad \forall (i,s) \in \mathcal{F}_a$$

$$x_{jtm} \in \{0,1\} \qquad \forall j \in \mathcal{J}, t \in \mathcal{W}_j$$

The formulation presented above will be modelled in Python and solved using the commercial solver Gurobi through Gurobipy. Given that the current problem is a an extension of the original NSWPP which is already an extension of the RCPSP, it is thus NP-hard will be more difficult to solve as the instance size increases. In what follows, we will present the adaptation made to the serial-SGS heuristic before the developed matheuristics are introduced.

3.4 Modified Serial-SGS Algorithm

The standard Serial Schedule Generation Scheme (Serial-SGS) is a heuristic approach used to generate a feasible schedule by considering the resources and time constraints of the project. The method involves generating a schedule by assigning activities to time slots in a serial manner, which means activities are assigned one by one in a pre-determined order. The objective of this method is to minimize the makespan, which is the total duration of the project (Chtourou & Haouari, 2008). The method uses a greedy approach to assign the activities to the earliest available slot and adjusts the schedule if resource conflicts arise. The key to serial-SGS is simply to schedule activities as early as possible. For the schedule generation process to work, there must be an ordered list of activities. The project activities are ordered according to various criteria such as priority, duration, and ES. With the help of these ranking rules, a workable schedule that front-load high-priority and lengthy tasks can be created.

A modified serial-SGS heuristic is needed to deal with multiple calendars, relationships and timings constraints which do not exist in the standard NSWPP or RCPSP. Although the adapted serial SGS seems complex with many steps, decision making, and formulations, its basic logic is to simply schedule activities as early as possible within resource availability. If timing constraints are not satisfied for an activity, the duration of its predecessors are iteratively reduced until a feasible schedule is obtained. The decision condition and statement $S_j = S_j + 1$ if $c_{[j,S_j]} = 0$ (line 50 and 51) is added to ensure that activities are not scheduled to start on a day-off. Without this condition, if the immediate predecessor ends on a day before a holiday, the successor will be scheduled to start on the holiday.

The pseudo-code of the modified serial-SGS developed by Yin (2022) is given below.

Algorithm 1: Compute the start time S_j of activity j

```
1: Input data: $n$, $d_{jtm}$, $db_{jtm}$, $M$, $R_{kt}$, $r_{jkt}$, $\mathcal{P}_{FS}$, $\mathcal{P}_{FF}$, $\mathcal{P}_{SS}$, $\mathcal{P}_{SF}$
2: Initialize: $j = 1$, $m_1 = 1$
3: while $j \leq n$ do
4:     Initialize $S_j = 1$
5:     if $\mathcal{P}_{FS} \neq \emptyset$ then
6:         Initialize: $i = 1$
7:         Find the cardinality $P_{FS}$ of set $\mathcal{P}_{FS}$: $P_{FS} = |\mathcal{P}_{FS}|$
8:         $S_{FS} = 1$
9:         while $i \leq P_{FS}$ do
10:            if $S_{FS} \leq S_i + d_{i,S_i,m_i}$ then
11:                $S_{FS} = Si + d_{i,S_i,m_i}$
12:            end if
13:            $i = i + 1$
14:        end while
15:    end if
16:    if $\mathcal{P}_{FF} \neq \emptyset$ then
17:        Initialize: $i = 1$
18:        Find the cardinality $P_{FF}$ of set $\mathcal{P}_{FF}$: $P_{FF} = |\mathcal{P}_{FF}|$
19:        $S_{FF} = 1$
20:        while $i \leq P_{FF}$ do
21:            if $S_{FF} \leq S_i + d_{i,S_i,m_i} - db_{j,S_i+d_{i,S_i,m_i},m_i}$ then
22:                $S_{FF} = S_i + d_{i,S_i,m_i} - db_{j,S_i+d_{i,S_i,m_i},m_i}$
```

```
23:         end if
24:         $i = i + 1$
25:       end while
26:   end if
27:   if $\mathcal{P}_{SS} \neq \emptyset$ then
28:       Initialize: $i = 1$
29:       Find the cardinality $P_{SS}$ of set $\mathcal{P}_{SS}$: $P_{SS} = |\mathcal{P}_{SS}|$
30:       $S_{SS} = 1$
31:       while $i \leq P_{SS}$ do
32:         if $S_{SS} \leq S_i$ then
33:             $S_{SS} = Si$
34:         end if
35:         $i = i + 1$
36:       end while
37:   end if
38:   if $\mathcal{P}_{SF} \neq \emptyset$ then
39:       Initialize: $i = 1$
40:       Find the cardinality $P_{SF}$ of set $\mathcal{P}_{SF}$: $P_{SF} = |\mathcal{P}_{SF}|$
41:       $S_{SF} = 1$
42:       while $i \leq P_{SF}$ do
43:         if $S_{SF} \leq S_i - db_{jS_im_j} + 1$ then
44:             $S_{SF} = S_i - db_{jS_im_j} + 1$
45:         end if
46:         $i = i + 1$
47:       end while
48:   end if
49:   $S_j = \max\{S_j, S_{FS}, S_{FF}, S_{SS}, S_{SF}\}$
50:   while $c_{jS_j} = 0$ do
51:       $S_j = S_j + 1$
52:   end while
53:   Initialize: g $= 0$
54:   while $g \leq d_{jS_jm_j}$ do
```

```
        if $R_{kS_j+g} < r_{jkS_j+g}$ then
            $S_j = S_j + g + 1$
            $g = 0$
        else
            $g = g + 1$
        end if
    end while
    if $S_j >$ Start-On $Date_j$ OR $S_j + D_{jS_jm_j} >$ Finish-On $Date_j$ OR $S_j >$ Start-On-or-Before $Date_j$ OR $S_j + D_{jS_jm_j} >$ Finish-On-or-Before $Date_j$ then
        $j = j - 1$
        while $m_j \geq M$ do
            $j = j - 1$
        end while
        $m_j = m_j + 1$
    else
        Initialize: $g = 0$
        while $g \leq D_{jS_jm_j}$ do
            $R_{S_j+g,k} = R_{S_j+g,k} - r_{j,k,S_j+g}$
            $g = g + 1$
        end while
        $j = j + 1$
        $m_j = 1$
    end if
end while
```

An example of this pre-processing is demonstrated in Tables 3.5 to 3.8. Activities, their priority, Earliest Start, and duration along with precedence relationship and timing constraints are demonstrated 3.5. Sorting the activities base on Priority, ES and Duration results in Table 3.6. Sorting the activities base on timing constraints results in the order of activities as demonstrated in 3.7. For the third step, the outcome of sorted activities is demonstrated in 3.8. 3.8 identifies the final order by which the activities are scheduled.

Table 3.5: Result of Standard Serial-SGS algorithm

Activity	Priority	ES	Duration	Precedence Relationship	Timing Constraint
1	1	0	5	Activity 3 (FS-Successor)	
2	1	3	1		Start on day 3
3	1	0	4	Activity 1 (FS-Predecessor)	
4	1	0	8		
5	2	0	4		
6	2	0	8		
7	1	7	2		Start on day 7
8	2	0	3		

Table 3.6: Results after Sorting by Priority, ES, and Duration

Activity	Priority	ES	Duration	Precedence Relationship	Timing Constraint
4	1	0	8		
1	1	0	5	Activity 3 (FS-Successor)	
3	1	0	4	Activity 1 (FS-Predecessor)	
2	1	3	1		Start on day 3
7	1	7	2		Start on day 7
6	2	0	8		
5	2	0	4		
8	2	0	3		

Table 3.7: Results after Sorting Based on Precedence Relationships

Activity	Priority	ES	Duration	Precedence Relationship	Timing Constraint
4	1	0	8		
3	1	0	4	Activity 1 (FS-Predecessor)	
1	1	0	5	Activity 3 (FS-Successor)	
2	1	3	1		Start on day 3
7	1	7	2		Start on day 7
6	2	0	8		
5	2	0	4		
8	2	0	3		

Table 3.8: Results after Sorting by Timing Constraints

Activity	Priority	ES	Duration	Precedence Relationship	Timing constraints
2	1	3	1		Start on day 3
7	1	7	2		Start on day 7
4	1	0	8		
3	1	0	4	Activity 1 (FS-Predecessor)	
1	1	0	5	Activity 3 (FS-Successor)	
6	2	0	8		
5	2	0	4		
8	2	0	3		

3.5 New Matheuristic Algoritms

In this section new developments to solve the NSWPP using a decomposition approach and a local search using improved versions of Multi-Step Optimization (MSO) are discussed. MSO being a matheuristic there is no guarantee of finding the optimal solution. However, it is hoped that it would find high-quality solutions in reasonable computation times. In a heuristic model, the goal is to find a local optimum. In a matheuristic model, the goal is to escape the local optimum and find a better local optimum in order to gradually approach the global optimum. In effect, Prabhu (2021) found the original MSO to efficiently provide close to optimal solutions than other heuristic methods.
Prabhu (2021) introduced a decomposition matheuristic where an instance with a large number of activities is solved by creating subgroups of the activities and then iteratively optimizing each subgroup using a binary integer programming model. Three variants (MSO–1, MSO–2, and MSO–3) were proposed. Brief descriptions of these variants were presented earlier in section 2.3.4.

In this thesis, two new variants of the MSO algorithm based on the agorithms proposed by Prabhu (2021) for solving the original NSWPP will be developed to handle the new features and requirements identified by our industry partners, *i.e.,* multi-calendar activities and resources, the three new activity relationships (FF, SS, and SF), and timing constraints as previously discussed. The new variants MSO–4 and MSO–5 are presented below.

3.5.1 Multi-Step Optimization Variant 4 (MSO–4)

The new variants must handle multi-calendar activities and resources, four activity relationships (FS, FF, SS, and SF), and timing constraints. MSO–4 can be summarized in four main steps as follows.

- **Step 1 – Data preparation**
  - Run Algorithm 1 and get an initial schedule satisfying all precedence relationships, multi-calendar and timing constraints. An example is depicted in Table 3.9.

Table 3.9: Result of Algorithm 1

Activity	Priority	ES	Duration	Precedence Relationship	Timing Constraint
1	1	0	5	Activity 3 (FS-Successor)	
2	1	3	1		Start on day 3
3	1	0	4	Activity 1 (FS-Predecessor)	
4	1	0	8		
5	2	0	4		
6	2	0	8		
7	1	7	2		Start on day 7
8	2	0	3		

- **Step 2 – Reordering the Activities**
  - First, activities are ordered based on three criteria (priority, ES, and duration) as shown in Table 3.10.
  - Second, activities with timing constraint (mandatory start date and end date) are moved at the top of the list, because they must be dealt with as early as possible when resources are available. See the example in Table 3.11.
  - Third, activities with precedence relationship are rearranged as required such that the successor activities are listed after their predecessors. See the example in Table 3.12 .

This sorting step ensures that activities are arranged in the desired order to satisfy the goal of front-loading high-priority and lengthy jobs while respecting other requirements.

Table 3.10: First Sorting Step for MSO–4

Sorting by Priority, ES, and Duration					
Activity	Priority	ES	Duration	Precedence Relationship	Timing Constraint
4	1	0	8		
1	1	0	5	Activity 3 (FS-Successor)	
3	1	0	4	Activity 1 (FS-Predecessor)	
2	1	3	1		Start on day 3
7	1	7	2		Start on day 7
6	2	0	8		
5	2	0	4		
8	2	0	3		

Table 3.11: Second Sorting Step for MSO–4

Sorting by Mandatory Start and End Dates					
Activity	Priority	ES	Duration	Precedence Relationship	Timing Constraint
2	1	3	1		Start on day 3
7	1	7	2		Start on day 7
4	1	0	8		
1	1	0	5	Activity 3 (FS-Successor)	
3	1	0	4	Activity 1 (FS-Predecessor)	
6	2	0	8		
5	2	0	4		
8	2	0	3		

- **Step 3 – Subgroup Decomposition**
  - Divide the ordered activity list into subgroups of predefined fixed size regardless of constraints and relationship requirements. Consequently, a subgroup may contain activities that have precedence relationships with activities in another subgroup. Each subgroup constitutes a sub-problem. The size of the subgroups can be decided by calibration or test-and-error. In this thesis, we used subgroups of size 50 after conducting a calibration experiment to be presented later in Chapter 4. Figure 3.1 Illustrates how

Table 3.12: Third Sorting Step for MSO–4

Sorting by Precedence Relationship					
Activity	Priority	ES	Duration	Precedence Relationship	Timing Constraint
2	1	3	1		Start on day 3
7	1	7	2		Start on day 7
4	1	0	8		
3	1	0	4	Activity 1 (FS-Predecessor)	
1	1	0	5	Activity 3 (FS-Successor)	
6	2	0	8		
5	2	0	4		
8	2	0	3		

subgroups are created under MSO–4 for subgroup size of two (2). Activities with the same colour are activities that have precedence relations with one another. Also note that if the total number of activities is not a multiple of the subgroup size, the last subgroup will contain less activities than the other subgroups.

- **Step 4 – Iterations**

  1. Optimize the scheduling problem using the BIP model for sub-problem 1 containing only subgroup 1.
  2. Using the results obtained in part 1, fix the start times and mode of execution of activities in sub-group 1. This is called "scheduling" activities in subgroup 1.
  3. Update the remaining resources availability by subtracting the resources used by the "scheduled activities" from the initial resource availability.
  4. Create a new sub-problem by adding the next subgroup to the current sub-problem (*i.e.*, the one that was just optimized). This is done to maintain precedence relationship and the timing constraints between sub-groups. See Figure 3.2.
  5. Repeat the sub-problem optimization, "scheduling subgroup activities, updating remaining resource availability, and adding the next subgroup to create a new sub-problem until all subgroups in the problem have been

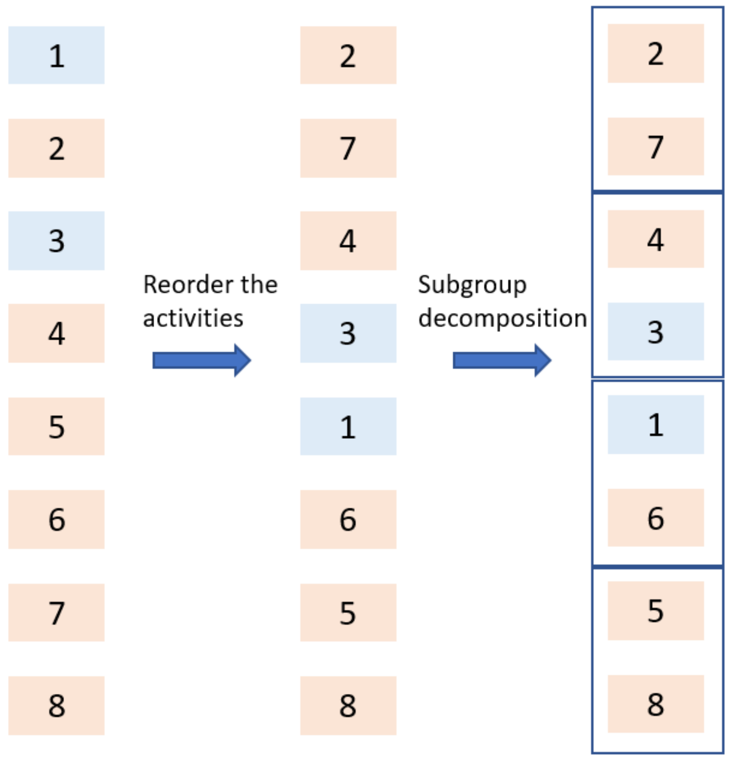

Figure 3.1: Illustration of subgroup creation under MSO–4

optimized and scheduled.

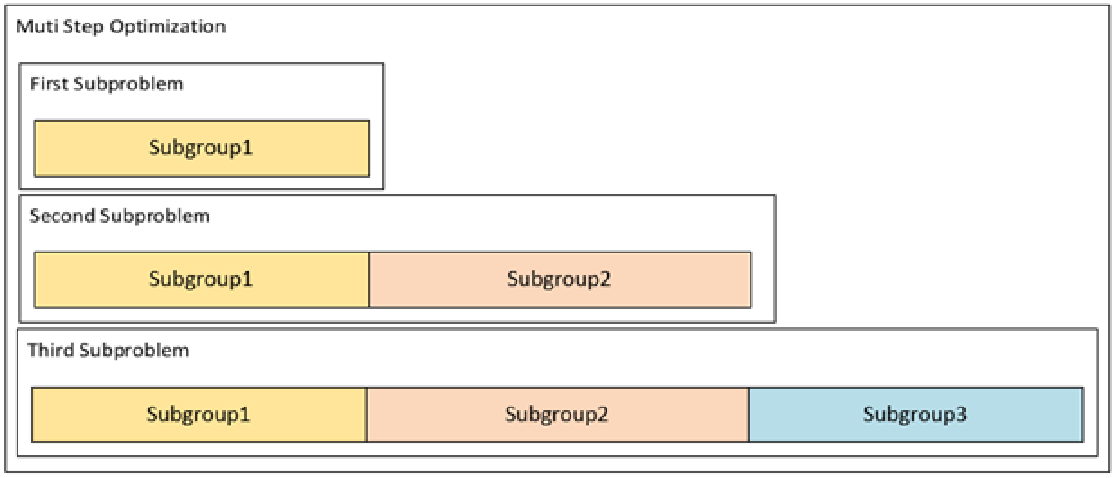

Figure 3.2: Illustration of subproblem creation under MSO–4

3.5.2 Multi-Step Optimization Variant 5 (MSO–5)

MSO–5 uses a different idea to create the subgroups. The decomposition in subgroups of mostly equal size in MSO–4 may result in precedence relationships between activities belonging to different subgroups. Thus, MSO–5 is designed to avoid these inter-subgroup dependencies by allowing subgroups to deviate from the pre-specified subgroup size. An activity along with all its “connected dependent activities” is placed in a *non-empty* subgroup, if adding them will not exceed the predefined subgroup size. Otherwise, they are placed in a new subgroup. An activity along with all its “connected dependent activities” is placed in an *empty* subgroup even if their number exceeds the predefined subgroup size (*i.e.,* they must be placed in a subgroup together).

For MSO–5, steps 1 and 2 are the same as for MSO–4. Step 3 creates the subgroups as described in the previous paragraph such that inter-subgroup dependencies are avoided. This then allows the iterative optimization (Step 4) to be performed on each individual subgroup as detailed below.

Step 4 – Iterations

1. Optimize the scheduling problem using the BIP model for sub-problem 1 (*i.e.*, also subgroup 1).

2. Using the results obtained in part 1, fix the start times and mode of execution of activities in sub-group 1.

3. Update the remaining resources availability by subtracting the resources used by the "scheduled activities" from the initial resource availability.

4. Repeat the optimization on the next subgroup, "scheduling subgroup activities, updating remaining resource availability, until all subgroups in the problem have been *individually* optimized and scheduled.

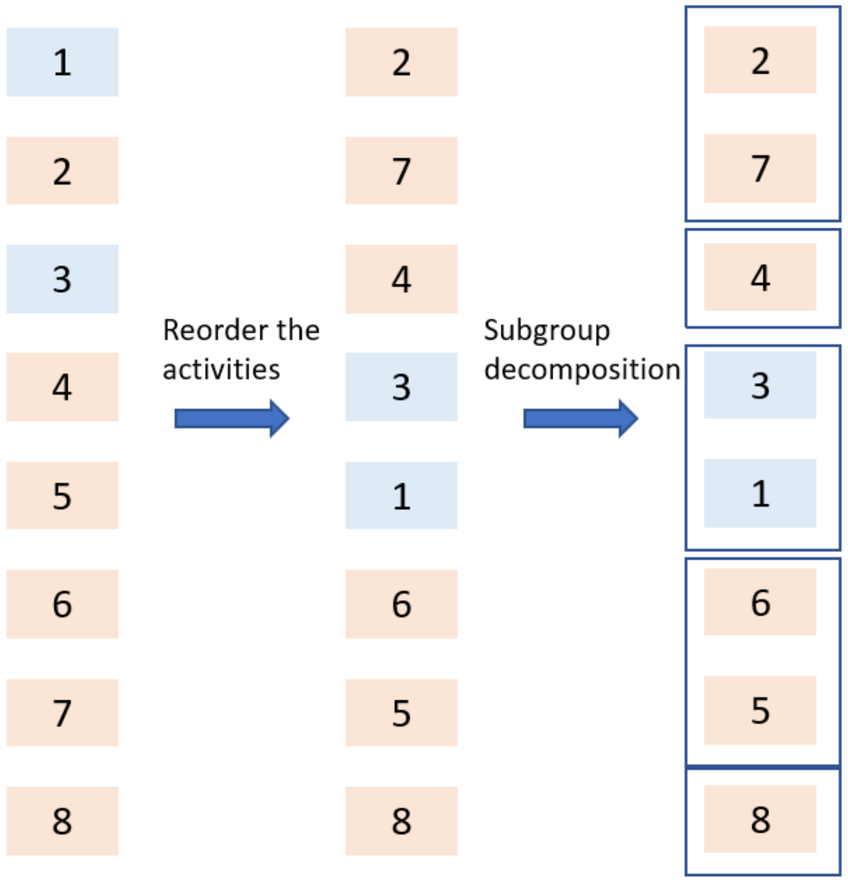

Figure 3.3: Illustration of subgroup creation under MSO–5

Figure 3.3 illustrates how subgroups are created under MSO–5 with a subgroup size of two (2). Activities 3 and 1 that are connected are now in the same subgroup unlike previously under MSO–4 where they were in two different subgroups (see Figure 3.1). Figure 3.4 illustrates the individual subgroups that are iteratively optimized under MSO–5.

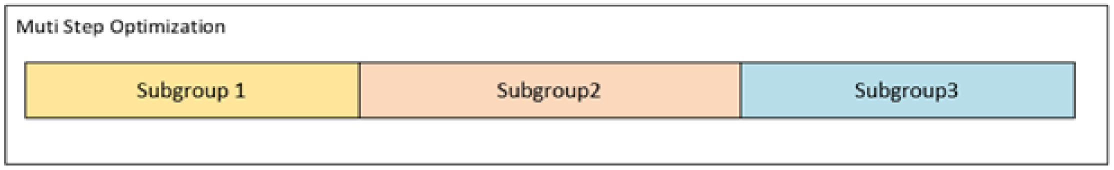

Figure 3.4: Illustration of subproblem creation under MSO–5

3.5.3 Weighted Average Sorting Criterion

The objective of this problem is to front-load activities with high priority and long duration. Thus, when ordering the activities before subgrouping them, they are sorted based on their priority, then their ES and finally on their duration. These are the sorting criteria used in the presentation of MSO–4 and MSO–5 above. Another criterion, the Average Weighted Sorting Score ($AWSS_j$) of each activity j as given by equation (3.15), will be used in the numerical experiments section. The parameters p_j, ES_j, and d_j are the priority, ES and duration of activity j respectively. This criterion would allow the decision maker to indicate their preferences by putting a weight (w_i) on each criterion i. The weights w_j, w_j, and w_j are weight given by the decision maker to the priority, ES and duration of activity j respectively.

$$AWSS_j = w_p\, p_j + w_{ES}\, ES_j + w_d\, d_j \qquad \forall j \in \mathcal{J} \tag{3.15}$$

where $w_p + w_{ES} + w_d = 1$. The performance of MSO-4 and MSO-5 are evaluated by implementing this sorting criteria as well.

Chapter 4

Numerical Experiments and Test Results

Several techniques were proposed to solve the NSWPP in the methodology section, including the BIP model, the modified serial-SGS heuristic, and two variant of a novel matheuristic MSO–4 and MSO–5 each with two sorting criteria. The decomposition method used in the heuristic and matheuristic methods is a combination of cutting and stacking (Sprecher, 2002). The algorithms are written in Python and are used to solve several examples to validate, verify, and compare the methodology introduced in this research. The BIP model is solved by Gurobi 9.5.1 from a code written in Pulp, an open-source linear integer programming modeller, which can call a variety of solvers (both open source and commercial). All computations were done on one of the Industrial Engineering Department Computational Servers (CPU clock rate: 2.20GHz; RAM: 16.00 GB).

4.1 Datasets and Instances

In this thesis, we use the OTC dataset with 136 activities called OTC–136 originally provided by our industrial partner. A 680-activity dataset called OTC–680 was generated by duplicating OTC-136 with the goal of having a more complex and difficult to solve problem (Yin, 2022). Table 4.1 shows the network complexity measures for these two instances. In the original OTC–136 dataset, all activities have the same priority level 3, which is not suitable to test the front-loading capabilities of the model and methods proposed. Thus, 6 new instances were created for each of OTC–136 and OTC–680 by randomly changing the priority levels of activities to 1 and 2. For OTC–136, 10 and 20 of the activities are randomly changed to priority 1 and 2 respectively. For OTC–680, 50 and 100 activities are randomly changed to priority 1 and 2 respectively. These new instances are identified by OTC–136/A and OTC–680/B where suffices A and B vary from 1 to 6 to represent each of the 6 new instances created.

Table 4.1: Network Indicators

Indicators	OTC–136	OTC–680
OS	0.0227	0.0045
CNC	0.7279	0.7279
RF	0.2312	0.2312
RC	0.4642	0.4642
PR	20.0000	20.0000
DR	0.3379	0.3379

4.2 Duration-Weighted Centroid (DWC)

Given that the NSWPP aims at front-loading high priority and long duration activities, a Duration Weighted Centroid (DWC) index was introduced by Bertrand (2020) to evaluate the quality of produced schedules by measuring the mean position of the center of mass of priority 1 activities. An improved DWC as shown in Equation (4.1) was proposed by Yin (2022). Equation 4.1 is intended to give more weight/importance to schedules that have long-duration activities starting early.

$$\text{DWC}=\sum_{i\in\mathcal{J}_1}\frac{S_i+F_i}{2\left|\mathcal{J}_1\right|}\,d_i^{1.1} \tag{4.1}$$

DWC is calculated by Equation 4.1, where $\mathcal{J}_1$ represents the set of priority-1 activities, S_i represent the start time of activities, F_i represent finish time of activity i respectively, and d_i represents the duration of activity i. DWC value is smaller when high-priority long-duration activities are front-loaded.

4.3 Choosing the Gap Tolerance for the BIP Optimization

The tolerance gap is the gap between the incumbent solution and the best known bound and it is used by optimization solvers as stopping criteria. If the tolerance gap is set too small, the solver may spent a considerable amount of time trying to reduce the optimization gap although the incumbent is optimal. Given that the NSWPP is a NP-hard problem and finding an exact solution for large-scale problems can be impossible in a reasonable amount of time, it can be beneficial to judiciously set the tolerance gap. Multiple runs of the BIP model introduced earlier were conducted on the OTC-680 dataset for various values of the tolerance gap and the obtained execution times (CPU_t), objective function value (Z), actual gap and DWC values were recorded as depicted in Table 4.2 and 4.3. A reasonable trade-off between the optimization gap and the computation time can be decided based on the result of these tables.

Table 4.2: Results for OTC–680

Tolerance (%)	CPU_t (s)	Z value	Z Gap(%)	DWC
BIP (0.10)	6966	2675	0.12	973.17
BIP (0.50)	2289	2678	0.25	974.11
BIP (1.00)	1085	2695	0.92	980.24
BIP (5.00)	619	2759	3.35	1003.12

Table 4.3: Results for OTC–680/2

Tolerance (%)	CPU_t (s)	Z value	Z Gap(%)	DWC
BIP (0.10)	5884	11452.3	0.259	1261.8
BIP (0.50)	3319	11461.7	0.540	1262.2
BIP (1.00)	1957	11490.8	0.963	1265.2
BIP (5.00)	271	11728.9	4.443	1277.4

As expected, the results in Tables 4.2 and 4.3 show that decreasing of the gap tolerance increases the computation time. The influence is more significant on larger datasets such as OTC–680 and OTC–680/2. The 0.1%, and 0.5% tolerance gaps lead to the best quality solutions in terms of Z value (objective value defined in Equation 3.1), actual gap and DWC. The best trade-off between accuracy of the solution and reasonable CPU time is obtained with the 1% tolerance gap. Therefore, the tolerance gap for the BIP model is set to 1% for the remainder of this thesis.

4.4 Selection and Calibration of Parameters for the Matheuristic Algorithms

Several parameters need to be set or calibrated before extensive comparative experiments can be carried out with the BIP model and MSO algorithms. Parameter selection can have substantial impact on solution quality and computation times. Some parameters may result in greater chance of generating better solutions. However, it can take a longer time to solve the problem. This section discusses the parameter values used by Bertrand (2020); Prabhu (2021); Yin (2022) and how several other parameters were selected or calibrated in the present study.

In the objective function (3.1), the term $(\varepsilon + d_{im})$ represents the duration of activity i under mode m to which ε, a very small number, is added to allow the modelling of zero-duration activities such as dummy activities or milestone activities which are commonly incurred in project management. The higher the values of exponent $\alpha > 1$ the more weight is given to the activity duration in the objective function. α is set to 1.1 and ε is set to 0.01 in this thesis as have done Bertrand (2020), Prabhu (2021), and Yin (2022).

The term p_i^{θ} in the denominator of the first term of the objective function ensures that activities with high priority are given higher weight and impact. The higher the value of θ, the higher the impact of the activity's priority on the objective function. Through extensive testing and to prevent scaling issues, Bertrand (2020) sets $\theta = 5$ as a reasonable value. This is the value that will be used in our experiments.

Parameter β is the exponent of the execution mode m in the objective function. As the value of β increases, the number of activities executed in modes higher than mode 1 will decrease. Recall that higher modes mean crashing activities through more resources and/or more overtime (*i.e.,* more costly). Thus, there is an incentive to choose an appropriate value for β. Multiple values were tested in order to determine the appropriate β for the problem under consideration. Table 4.4 shows the results obtained. A good and reasonable range for β is between 8 and 15 as its offers a good trade-off between computation time and optimality gap as well as not selecting higher execution modes. For the remainder of the tests in this thesis, $\beta = 10$ is selected as it gives the best combination of computation time, optimality gap, centroid, and makespan.

Table 4.4: Results for Different Values of Parameter β

						Number of activities		
β	CPU_t	Z	Z Gap(%)	DWC	Makespan	Mode 1	Mode 2	Mode 3
5	9322	2674.11	0.67	966.4	567	658	22	0
6	9603	2692.23	0.87	975.5	567	672	8	0
7	9903	2679.74	0.33	974.7	567	680	0	0
8	8864	2695.61	0.92	980.2	571	680	0	0
9	8743	2695.61	0.92	980.2	571	680	0	0
10	8677	2676.54	0.25	973.6	567	680	0	0
15	8776	2684.58	0.54	976.3	571	680	0	0
30	9241	2696.54	0.96	980.6	578	680	0	0

The values of θ, α, β, and ε_1 are set to 5, 1.1, 10, and 0.001 respectively. Thus, the objective function used to solve the problem is:

$$\text{Minimize}\, Z = \sum_{i\in\mathcal{J}} \sum_{t\in\mathcal{W}_i} \sum_{m\in\mathcal{M}} \frac{x_{itm}}{p_i^5} \left(0.001 + d_{im}\right)^{1.1} \cdot t \cdot m^{10}$$

4.5 Comparison Between Minimization, Maximization models, and Serial-SGS

In this thesis, the developed BIP formulation is a minimization. A maximization formulation with the same constraints was proposed by Yin (2022). A study is carried out to compare performance of the Minimization model, Maximization model, and the Serial SGS heuristic. The comparison is based on the DWC and makespan resulting from the solutions obtained. The problem is run on 5 instances obtained by duplicating the original OTC–136 dataset for up to 4 times as indicated in Table 4.5.

The results shown in Table 4.5 indicate that the minimization model has lower

Table 4.5: Minimization, Maximization formulations and Serial-SGS Comparison

	Minimization		Maximization		SGS	
Size (n)	DWC	Makespan	DWC	Makespan	DWC	Makespan
136	**235.5**	180	290.4	180	320.8	181
272	**403.5**	231	534.3	**229**	567.5	249
408	**591.4**	343	762.7	**340**	854.4	382
544	**784.6**	462	1015.7	**455**	1094.0	511
680	**980.2**	**571**	1277.8	574	1340.6	630

DWC in all instances, meaning that it does a better job front-loading priority-1 activities. The maximization model produces schedules with shorter or equal makespan in most cases. Both minimisation and maximization formulations produce better solutions in term of DWC and makespan than the Serial-SGS.

4.6 Impact of Subgroup Size for MSOs

MSO–4 and MSO–5 introduced in this thesis, slice the original activity set to subgroups of activities and iteratively solve each subgroup or a combination of subgroups until the original problem is solved. Subgroup size must be carefully calibrated. An experiment is carried out to illustrate how such a decision could be made. The OTC–136 and OTC–680 instance were solved by the MSO–4 algorithm using subgroup sizes varying from 2 to 100. The obtained results are depicted in Tables 4.6 and 4.7 in terms of Data Preparation Time, Constraint Generation Time, and Solver CPU Time, as well as Objective Value (Z), Makespan ($C_{\max}$) and DWC.

Data Preparation Time is the time it takes to read the data from the raw data files and converting them into a format that PuLP and Python codes can read. Constraint Generation Time is the time needed to construct the problem's constraints in PuLP. Solver solution time is the time required by the solver CPU_t is the time used by Gurobi to solve the problem. Total Time is the sum of all previous times.

Table 4.6: Average performance of MSO–4 on OTC–136 with 1% optimality tolerance

Subgroup Size	Data Prep. time (s)	Constraint Generat. time (s)	Solver CPU_t (s)	Total time (s)	Z	$C_{\max}$	DWC
2	34.7	468.73	00.62	504.05	123.48	180	240.23
3	33.4	465.76	00.43	499.59	123.48	180	240.23
5	34	455.13	00.52	489.65	124.21	180	242.02
10	33	462.51	00.76	496.27	128.06	180	248.21
20	34	460.37	01.83	496.20	125.16	180	243.48
30	34	456.46	02.52	492.98	126.98	180	246.37
50	34	462.64	02.81	499.45	124.19	180	242.27
70	34	462.83	03.93	500.76	127.30	180	247.25
100	33	459.19	08.49	500.68	124.32	180	242.56

Table 4.6 displays the results obtained with OTC–136. It shows that while problem Data Preparation times, Constraint Generation times and Total Times are not significantly impacted by subgroup size, the Solver CPU_t increases when the subgroup size increases. The objective value Z, DWC and makespan are not significantly affected by the subgroup size. The fact that the dataset size is small may explain why the subgroup size is not a significant factor in this case.

For OTC–680, Table 4.7 shows similar trends as with the smaller dataset. While Data Preparation times and Constraint Generation times are not impacted by subgroup size, the Solver CPU_t increases when the subgroup size increases. There is negligible variation for Z and DWC across the range of subgroup size. Subgroup size 30 exhibits an interesting result where the Solver CPU_t and Total time are substantially lower at 40.6 seconds and 9619 seconds respectively. It is the subgroup that offers the best compromise between computation time and objective function, and will be used for the remaining numerical experiments.

Table 4.7: Average performance of MSO–4 on OTC–680 with 1% optimality tolerance

Subgroup Size	Data Prep. time (s)	Constraint Generat. time (s)	Solver CPU_t (s)	Total time (s)	Z	$C_{\max}$	DWC
2	706	9106	14.1	9872	14854.0	588	1110.9
3	708	9082	12.9	9909	14751.7	595	1103.9
5	706	8976	15.7	9810	14651.0	630	1110.8
10	711	9026	25.3	9951	14648.2	630	1119.6
20	710	8931	44.9	9890	14620.2	623	1119.7
30	714	9031	40.6	9619	14608.7	630	1120.8
50	707	8950	48.8	9925	14626.4	630	1120.1
70	706	9045	97.4	9903	13923.7	630	1123.9
100	710	9374	130.2	9893	14388.2	616	1110.2

4.7 Comparison of SGS, BIP and MSO

This section presents a comparison of the Serial-SGS, BIP (with different optimality gaps), MSO–4 and MSO–5 algorithms on computation times and solution quality. No time limits were imposed for the different runs of the BIP model. They were run until an optimum solution was achieved. The parameter combinations derived from the previous calibration sections are applied for this experiment. Tables 4.8 and Table 4.9 display the results obtained for OTC–136 and OTC–680 respectively.

Table 4.8: Results for OTC–136

	Data Gener. Time(s)	Constr. Prep. Time(s)	BIP CPU_t(s)	Total Time(s)	Z	Opt. Gap (%)	$C_{\max}$	DWC
SGS	–	–	–	**30**	2418.3	19.1	181	328
BIP(0.1%)	33.1	439.0	5.18	378	2055.0	0.1	180	268
BIP(0.5%)	33.0	438.7	4.31	377	2058.0	0.3	180	272
BIP(1%)	34.8	437.9	4.04	377	2061.8	0.5	180	273
BIP(5%)	33.3	438.4	3.38	377	2100.2	3.9	180	312
MSO–4	33.4	447.0	**2.53**	491	2054.4	1.7	180	264
MSO–5	33.3	493.0	4.48	494	3034.1	47.7	180	260

Table 4.9: Results for OTC–680

	Data Gener. Time(s)	Constr. Prep. Time(s)	BIP CPU_t(s)	Total Time(s)	Z	Opt. Gap (%)	$C_{\max}$	DWC
SGS	–	–	–	**649**	16095	44.1	630	1420
BIP(0.1%)	686.9	6971.8	6971	14119	11452	0.3	588	1262
BIP(0.5%)	702.3	6917.7	2296	10475	11462	0.5	571	1262
BIP(1%)	706.2	6947.7	1090	9330	11491	1.0	567	1265
BIP(5%)	699.4	6974.0	632	8132	11729	4.4	567	1277
MSO–4	701.6	8806.0	**43**	9695	11954	6.5	630	1136
MSO–5	707.9	9658.0	66	10126	63932	479.0	630	1218

It is evident from Tables 4.8 and 4.9 that the Total Time of the Serial-SGS is significantly smaller than any other method. This is expected as this is a crude although efficient heuristic and it is also a required step for all other methods. The data preparation times are consistent between the BIP model and MSO algorithms. Small differences observed can be traced back to server workload variations and other unknown factors.

Constraints generation times are consistent for the four BIP versions as the optimality gap is not involved at this stage. Constraints generation times are slightly higher for MSO–5 than MSO–4 due to the additional steps needed when creating subgroups for MSO–5. For the BIP models, the constraints are generated by PuLP an open-source package for which development has stopped. It is expected that using newer LP modellers such as Gurobipy would bring significant improvements to this phase.
For the MSO algorithms the constraint generation was coded in Python. A code written by a more advanced-coder could potentially reduce these times and bring the MSO algorithms to the level of PuLP.

In terms of pure Solver times, MSO–4 and MSO–5 produce the lowest times. This is more evident with the results achieved for the OTC–680 dataset where MSO only needs 43 seconds. In terms of total times, BIP (5%) , BIP (1%) and MSO–4 perform well. BIP (5%) has the lowest total time at 8,132 seconds for a 4.4% optimality gap. BIP (1%) takes 9,330 seconds to reach a 1% optimality gap. MSO–4 9,695 seconds to reach a 6.5% optimality gap. MSO–4 gets the best DWC at 1136 whereas BIP (1%) and BIP (5%) get 1265 and 1277 respectively. MSO–4 is capable of better frontloading activities than the BIP models. It should be noted that the objective function of BIP model is not DWC. Although MSO–5 takes more time than the BIP (1%), BIP (5%) and MSO–4, it still achieved a better DWC than BIP (1%) and BIP (5%).

In summary, BIP (5%) , BIP (1%) and MSO–4 perform well. The matheuristic principle based on subgroup decomposition is promising. It needs better coding implementation in Python and use newer modellers such as Gurobipy to decrease the constraint generation time to make the MSO–4 and MSO–5 more efficient.

4.8 Comparing the Sorting Methods

As explained in subsection 3.5.3, when ordering the activities before subgrouping them, they are sorted based on their priority, then their ES and finally on their duration. This is a sequential sorting method with no weight given to the sorting criteria herein denoted "Sequential–no weight". Another criterion is the Average Weighted Sorting Score ($AWSS_j$) of each activity j as given by equation (3.15). This section compares the results obtained with MSO–4 using the "Sequential-no weight", and the Average Weighted Sorting Score ($AWSS_j$) with different weight schemes resulting in 12 runs for each of the OTC–136/1 and OTC680–/1 datasets.

The first run uses the "sequential–no weight" method. Runs 2 to 7 use the weighted average score with Priority having higher weight than ES and Duration, and then ES having higher weight than Duration. Runs 8 to 12, use the weighted average score with Priority having higher weight than ES and Duration, and then Duration having higher weight than ES. The comparison of Total time, Z value, optimality gap, DWC and makespan is displayed in Tables 4.10 and 4.11 for OTC–186/1 and OTC–680/1 respectively.

As expected, the sorting method has no bearing on the total computation times. However, the observed results show that the weighted average method outperforms the sequential method. An empirical observation is that runs 2 to 7 seem to produce better results than runs 8 to 12. A strong recommendation is then to use the weighted average score to order the activities before subgrouping them. A weaker recommendation is to use the weighted average score with Priority having higher weight than ES and Duration, and ES having higher weight than Duration. Additional results obtained with OTC–136/2 and OTC–680/2 reported below in Tables 4.12 and 4.13 respectively confirm the previous observations. Further investigations are needed to confirm the empirical observations made here.

Table 4.10: Comparison of the Two Sorting Methods on OTC–136/1

Runs	Weights (Priority, ES, Duration)	Total Time(s)	Z	Gap (%)	DWC	$C_{\max}$
1	(Sequential–no weight)	483	3262.5	4.68	291.0	181
2	(0.90, 0.05, 0.05)	486	3128.5	0.38	258.7	180
3	(0.85, 0.10, 0.05)	484	3128.0	0.36	259.9	180
4	(0.75, 0.15, 0.10)	483	3129.3	0.40	260.3	180
5	(0.70, 0.20, 0.10)	480	3127.9	0.36	259.9	180
6	(0.55, 0.25, 0.20)	487	3128.2	0.37	258.0	180
7	(0.50, 0.30, 0.20)	482	3128.2	0.37	258.1	180
8	(0.85, 0.05, 0.10)	488	3128.8	0.39	259.3	180
9	(0.75, 0.10, 0.15)	487	3128.6	0.38	258.8	180
10	(0.70, 0.10, 0.20)	491	3128.8	0.39	259.3	180
11	(0.55, 0.20, 0.25)	490	3128.6	0.38	258.8	180
12	(0.50, 0.20, 0.30)	486	3128.0	0.36	257.7	180

Table 4.11: Comparison of the Two Sorting Methods on OTC–680/1

Runs	Weights (Priority, ES, Duration)	Total Time(s)	Z	Gap (%)	DWC	$C_{\max}$
1	(Sequential–no weight)	9917	14557.1	6.00	1050.5	580
2	(0.90, 0.05, 0.05)	9899	13788.2	0.11	1039.8	567
3	(0.85, 0.10, 0.05)	9823	13812.6	0.28	1047.2	588
4	(0.75, 0.15, 0.10)	9748	13799.1	0.18	1041.5	581
5	(0.70, 0.20, 0.10)	9852	13805.9	0.23	1045.7	588
6	(0.55, 0.25, 0.20)	9902	13795.4	0.16	1042.3	581
7	(0.50, 0.30, 0.20)	9932	13797.7	0.17	1042.8	581
8	(0.85, 0.05, 0.10)	9903	13796.4	0.17	1045.6	581
9	(0.75, 0.10, 0.15)	9865	13811.3	0.27	1049.0	567
10	(0.70, 0.10, 0.20)	9835	13796.4	0.17	1045.6	581
11	(0.55, 0.20, 0.25)	9826	13816.7	0.31	1045.4	574
12	(0.50, 0.20, 0.30)	9859	13844.1	0.51	1041.5	574

Table 4.12: Comparison of the Two Sorting Methods on OTC–136/2

Runs	Weights (Priority, ES, Duration)	Total Time(s)	Z	Gap (%)	DWC	$C_{\max}$
1	(Sequential–no weight)	491	2190.7	0.90	279.5	180
2	(0.90, 0.05, 0.05)	480	2192.5	0.99	264.4	180
3	(0.85, 0.10, 0.05)	481	2192.6	0.99	264.5	180
4	(0.75, 0.15, 0.10)	482	2192.4	0.98	264.5	180
5	(0.70, 0.20, 0.10)	477	2192.5	0.99	264.6	180
6	(0.55, 0.25, 0.20)	478	2191.9	0.96	264.0	180
7	(0.50, 0.30, 0.20)	471	2192.3	0.98	264.7	180
8	(0.85, 0.05, 0.10)	484	2197.5	1.22	266.9	180
9	(0.75, 0.10, 0.15)	481	2191.7	0.95	262.8	180
10	(0.70, 0.10, 0.20)	480	2197.5	1.22	266.9	180
11	(0.55, 0.20, 0.25)	485	2192.4	0.98	264.4	180
12	(0.50, 0.20, 0.30)	474	2192.9	1.01	265.1	180

Table 4.13: Comparison of the Two Sorting Methods on OTC–680/2

Runs	Weights (Priority, ES, Duration)	Total Time(s)	Z	Gap (%)	DWC	$C_{\max}$
1	(Sequential–no weight)	9942	8807.4	6.74	1140.7	637
2	(0.90, 0.05, 0.05)	9707	8290.0	0.92	1052.6	637
3	(0.85, 0.10, 0.05)	9611	8330.0	1.40	1047.2	637
4	(0.75, 0.15, 0.10)	9658	8261.0	0.58	1052.6	637
5	(0.70, 0.20, 0.10)	9714	8330.0	1.40	1047.2	637
6	(0.55, 0.25, 0.20)	9685	8296.4	1.01	1052.8	637
7	(0.50, 0.30, 0.20)	9593	8269.0	0.67	1055.1	637
8	(0.85, 0.05, 0.10)	9615	8287.7	0.90	1062.2	637
9	(0.75, 0.10, 0.15)	9581	8269.2	0.68	1056.2	637
10	(0.70, 0.10, 0.20)	9610	8287.7	0.90	1062.2	637
11	(0.55, 0.20, 0.25)	9587	8288.6	0.91	1055.5	637
12	(0.50, 0.20, 0.30)	9665	8638.4	4.92	1049.0	637

4.9 Comparing the BIP Models and MSO–4 with Weighted Sorting

Table 4.14 compares the results obtained using the BIP model with tolerance gaps of 0.1% and 1%, and 6 runs of MSO–4 using the weighted average score for sorting activities on OTC–680/2. The results show that MSO–4 is competitive in terms of frontloading and computation times despite its "unoptimized Python coding" implementation. BIP with 1% tolerance is still the overall best. MSO–4 (0.75, 0.15, 0.10) offers a good compromise between computation time, Z value and DWC.

Table 4.14: Comparison of BIP and MSO–4 with Weighted Sorting on OTC–680/2

Models/Algorithms	Total Time (s)	Z	Gap (%)	DWC	$C_{\max}$
BIP (0.1%)	14647	8221.7	0.10	1236.0	637
BIP (1%)	8287	8256.6	0.52	1242.8	637
MSO–4 (0.85, 0.10, 0.05)	9611	8330.0	1.40	1047.2	637
MSO–4 (0.75, 0.15, 0.10)	9658	8261.0	0.58	1052.6	637
MSO–4 (0.70, 0.20, 0.10)	9714	8330.0	1.40	1047.2	637
MSO–4 (0.50, 0.30, 0.20)	9593	8269.0	0.67	1055.1	637
MSO–4 (0.75, 0.10, 0.15)	9581	8269.2	0.67	1056.2	637
MSO–4 (0.50, 0.20, 0.30)	9587	8288.6	0.91	1055.5	637

Chapter 5

Conclusion and Discussion

This thesis proposed a new formulation and two matheuristic algorithms to deal with the multi-calendar naval surface ship work period problem, which involves the scheduling of maintenance tasks on naval ships with complex operational constraints such as multiple execution modes, and timing constraints. First, a binary integer programming model was developed for the NSWPP. Then, two novel matheuristic algorithms that combine mathematical programming with heuristic techniques were designed. The methods were tested on two sets of instances inspired by real-life naval vessel refit operations and compared to other methods in the literature.

Numerical experiments showed that the proposed BIP model could achieve optimal solutions with Gurobi in a reasonable amount of time for moderate-size instances (under 400 activities) with optimality gap reasonabily set at 1%. However, for large-size instances, the solver takes a very long to find the optimal solution. The proposed MSO algorithms showed potential for finding high-quality solutions but the code efficiency of the constraint generation phase must be improved. Empirical observations from some of the numerical experiments suggest using the weighted average score to order the activities before subgrouping them. A weaker recommendation is to use the weighted average score with Priority having higher weight than ES and Duration, and ES having higher weight than Duration.

Overall, the proposed method represents a significant contribution to the field of naval operations research. It provides a powerful tool for scheduling maintenance tasks on naval ships, which can lead to increased efficiency, reduced downtime, and improved operational readiness. Furthermore, the matheuristic approach can be applied to other scheduling problems in various domains, making it a valuable tool for operations research more broadly.

There are several potential directions for future research on the multi-calendar naval surface ship work period problem and the proposed matheuristic method. A few suggestions are described below.
The first research avenue is the thorough investigation of the weighted average sorting score to order the activities before subgrouping them. Empirical observations suggest that it is superior to the sequential Priority-ES-Duration method. Extensive experiments with statistical tests would allow to define the best combination of weights to use small, moderate and large instances of the NSWPP.

The second research avenue would be to explore other types of formulations for the multicalendar NSWPP. Indeed, so far only the discrete-time formulation has been used. Flow-based formulations, Event On/Off formulations and even Constraint Programming (CP) have not been used. Recent developments in CP could allow for efficient algorithms.

Finally, it would be interesting to use recent advances in machine learning to tackle the optimization of the NSWPP. For example, reinforcement learning (RL) is a very promising avenue for combinatorial optimization research because of its efficacy in terms of solution quality, ability to beat existing algorithms, and large running time advantages over traditional heuristic techniques (Mazyavkina et al., 2021; Zhang et al., 2022).

References

Afshar-Nadjafi, B. (2021). Multi-skilling in scheduling problems: A review on models, methods and applications. *Computers & Industrial Engineering*, *151*, 107004.

Alcaraz, J., Maroto, C., & Ruiz, R. (2003). Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. *Journal of the Operational Research Society*, *54*(6), 614–626.

Alipouri, Y., Sebt, M. H., Ardeshir, A., & Zarandi, M. H. F. (2020). A mixed-integer linear programming model for solving fuzzy stochastic resource constrained project scheduling problem. *Operational Research*, *20*, 197–217.

An, D., Parragh, S. N., Sinnl, M., & Tricoire, F. (2021). A lp relaxation based matheuristic for multi-objective integer programming. *arXiv preprint arXiv:2102.03582*.

Artigues, C., Michelon, P., & Reusser, S. (2003). Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, *149*(2), 249–267.

Ballestín, F., Valls, V., & Quintanilla, S. (2006). Due dates and rcpsp. *Perspectives in modern project scheduling*, 79–104.

Bertrand, E. (2020). *Optimization of the naval surface ship resource-constrained project scheduling problem* (Unpublished master's thesis). Dalhousie University.

Bistarelli, S., Frühwirth, T., Marte, M., & Rossi, F. (2004). Soft constraint propagation and solving in constraint handling rules. *Computational intelligence*, *20*(2), 287-307.

Blazewicz, J., Lenstra, J. K., & Rinnooy, A. K. (1983). Scheduling subject to resource constraints: classification and complexity. *Discrete applied mathematics*, *5*(1), 11–24.

Boer, R., J.M.J.Schutten, & W.H.M.Zijm. (1997). A decision support system for ship maintenance capacity planning. *CIRP Annals*, *46*(1), 391–396.

Chakrabortty, R. K., Sarker, R. A., & Essam, D. L. (2020). Single mode resource constrained project scheduling with unreliable resources. *Operational Research*, *20*, 1369–1403.

Chen, R. (2002). *An optimal control based plantwide control design methodology and its applications.* University of Maryland, College Park.

Cho, J., & Kim, Y.-D. (1997). A simulated annealing algorithm for resource constrained project scheduling problems. *Journal of the Operational Research Society*, *48*(7), 736–744.

Christofides, N., Alvarez-Valdés, R., & Tamarit, J. M. (1987). Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, *29*(3), 262–273.

Chtourou, H., & Haouari, M. (2008). A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling. *Computers & industrial engineering*, *55*(1), 183–194.

Coelho, J., & Vanhoucke, M. (2014). *Multi-mode resource-constrained project scheduling using rcpsp and sat solvers* (Vol. 1). Springer.

Cooper, D. F. (1976). Heuristics for scheduling resource-constrained projects: An experimental investigation. *Management Science*, *22*(11), 1186–1194.

Couch, J. (2016). *Performance robust project scheduling policies for naval ship maintenance* (Unpublished master's thesis). Dalhousie University.

Daniels, R. L., Hoopes, B. J., & Mazzola, J. B. (1996). Scheduling parallel manufacturing cells with resource flexibility. *Management science*, *42*(9), 1260–1276.

de Azevedo, G. H. I., Pessoa, A. A., & Subramanian, A. (2021). A constraint programming approach for the resource-constrained project scheduling problem. *European journal of operational research*, *289*(3), 809-824.

Demeulemeester, E., & Herroelen, W. (1992). A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management science*, *38*(12), 1803–1818.

Demeulemeester, E., Herroelen, W., Simpson, W. P., Baroum, S., Patterson, J. H., & Yang, K.-K. (1994). On a paper by christofides et al. for solving the multiple-resource constrained, single project scheduling problem. *European Journal of Operational Research*, *76*(1), 218–228.

De Nijs, F. (2013). *Project scheduling: The impact of instance structure on heuristic performance* (Unpublished master's thesis). Technical University Delft.

Dridi, O., Krichen, S., & Guitouni, A. (2012). A multi-objective optimization approach for resource assignment and task scheduling problem: Application to maritime domain awareness. *IEEE Electronic Library (IEL) Journals*, 646–669.

F. Rossi, P. v. B., & Walsh, T. (2006). *Handbook of constraint programming.* Elsevier.

Gomes, Castilho, M., Póvoa, A. P. B., & Novais, A. Q. (2010). A discrete time reactive scheduling model for new order insertion in job shop, make-to-order industries. *Computers & chemical engineering*, *48*(24), 7395–7422.

Günther, H. M. G., & Neuhaus, U. (2006). Realizing block planning concepts in make-and-pack production using milp modelling and sap apo. *International journal of production research*, *44*(18–19), 3711-3726.

Harjunkoski, I., & Grossmann, I. E. (2002). Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. *Computers & chemical engineering*, *26*(11), 1533–1552.

Hartmann, S. (1998). A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics (NRL)*, *45*(7), 733–750.

Herroelen, W., & De Reyck, B. (1999). Phase transitions in project scheduling. *Journal of the Operational Research Society*, *50*(2), 148–156.

Hooker, J. N. (2002). Logic, optimization and constraint programming. *INFORMS journal on computing*, *14*(4), 295–321.

Jarboui, B., Damak, N., Siarry, P., & Rebai, A. (2008). A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. *Applied Mathematics and Computation*, *195*(1), 299–308.

Jiang, G., & Shi, J. (2005). Exact algorithm for solving project scheduling problems under multiple resource constraints. *Journal of construction engineering and management*, *131*(9), 986–992.

Józefowska, J., Mika, M., Różycki, R., Waligóra, G., & Weglarz, J. (2001). Simulated annealing for multi-mode resource-constrained project scheduling. *Annals of Operations Research*, *102*, 137–155.

Kelley, & E, J. (1963). The critical-path method: resource planning and scheduling. *Industrial scheduling*.

Kelley, J. J. E., & Walker, M. R. (1959). Critical-path planning and scheduling. In *Papers presented at the december 1-3, 1959, eastern joint ire-aiee-acm computer conference* (pp. 160–173).

Kerzner, H. (2017). *Project management: a systems approach to planning, scheduling, and controlling*. Hoboken, New Jersey: Wiley.

Kolisch, R. (2013). *Project scheduling under resource constraints: efficient heuristics for several problem classes*. Springer Science & Business Media.

Kolisch, R., & Hartmann, S. (1999). *Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis*. Springer.

Kolisch, R., & Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European journal of operational research*, *174*(1), 23–37.

Kolisch, R., Sprecher, A., & Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Management science*, *41*(10), 1693–1703.

Koné, O., Artigues, C., Lopez, P., & Mongeau, M. (2011). Event-based milp models for resource-constrained project scheduling problems. *Computers & Operations Research*, *38*(1), 3–13.

Koné, O., Artigues, C., Lopez, P., & Mongeau, M. (2013). Comparison of mixed integer linear programming models for the resource-constrained project scheduling problem with consumption and production of resources. *Computers & Operations Research*, *25*(1-2), 25–47.

Kopanos, G. M., Méndez, C. A., & Puigjaner, L. (2010). Mip-based decomposition strategies for large-scale scheduling problems in multiproduct multistage batch plants: A benchmark scheduling problem of the pharmaceutical industry. *Industrial & engineering chemistry research*, *207*(2), 644–655.

Kopanos, G. M., Puigjaner, L., & Georgiadis, M. C. (2012). Efficient mathematical frameworks for detailed production scheduling in food processing industries. *Computers & chemical engineering*, *42*, 206–216.

Li, C., Zhang, Y., Su, X., & Wang, X. (2022). An improved optimization algorithm for aeronautical maintenance and repair task scheduling problem. *Mathematics*, *10*(20), 3777.

Li, H., Xiong, L., Liu, Y., & Li, H. (2018). An effective genetic algorithm for the resource levelling problem with generalised precedence relations. *International journal of production research*, *56*(5), 2054–2075.

Malcolm, D. G., Roseboom, J. H., Clark, C. E., & Fazar, W. (1959). Application of a technique for research and development program evaluation. *Operations research*, *7*(5), 646–669.

Maniezzo, V., Stützle, T., & Voß, S. (2021). *Matheuristics*. Springer.

Maravelias, C. T. (2006). A decomposition framework for the scheduling of single-and multi-stage processes. *Computers & Chemical Engineering*, *30*(3), 407–420.

Mazyavkina, N., Sviridov, S., Ivanov, S., & Burnaev, E. (2021). Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, *134*, 105400.

Méndez, C., & Cerdá, J. (2002). An milp-based approach to the short-term scheduling of make and pack continuous production plants. *Computers & chemical engineering*, *24*(4), 403–429.

Méndez, C., & Cerdá, J. (2003). Dynamic scheduling in multiproduct batch plants. *Computers & chemical engineering*, *27*(8), 1247-1259.

Merkle, D., Middendorf, M., & Schmeck, H. (2002). Ant colony optimization for resource-constrained project scheduling. *IEEE transactions on evolutionary computation*, *6*(4), 333–346.

Neumann, K., Schwindt, C., & Trautmann, N. (2002). Advanced production scheduling for batch plants in process industries. *OR spectrum*, *24*, 251–279.

Nguyen, V. H. (2017). *Optimal ship maintenance scheduling under restricted conditions and constrained resources* (Unpublished master's thesis). Hanoi University of Mining and Geology.

Patterson, J. H. (1976). Project scheduling: The effects of problem structure on heuristic performance. *Naval Research Logistics Quarterly*, *23*(1), 95–123.

Pellerin, R., Perrier, N., & Berthaut, F. (2020). A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, *280*(2), 395–416.

Prabhu, S. M. (2021). *Matheuristic optimization approaches for large scale resource constrained scheduling problems in refit operations* (Unpublished master's thesis). Dalhousie University.

Pritsker, A. A. B., Waiters, L. J., & Wolfe, P. M. (1969). Multiproject scheduling with limited resources: A zero-one programming approach. *Management science*, *16*(1), 93–108.

Roslöf, J., Harjunkoski, I., Björkqvist, J., Karlsson, S., & Westerlund, T. (2001). An milp-based reordering algorithm for complex industrial scheduling and rescheduling. *Computers & Chemical Engineering*, *25*(4–6), 821–828.

Roslöf, J., Harjunkoski, I., Westerlund, T., & Isaksson, J. (2002). Solving a large-scale industrial scheduling problem using milp combined with a heuristic procedure. *European Journal of Operational Research*, *138*(1), 29–42.

Sathi, A., Fox, M. S., & Greenberg, M. (1985). Representation of activity knowledge for project management. *IEEE transactions on pattern analysis and machine intelligence*, *7*(5), 531–552.

Schnell, A., & Hartl, R. F. (2017). On the generalization of constraint programming and boolean satisfiability solving techniques to schedule a resource-constrained project consisting of multi-mode jobs. *Operations Research Perspectives*, *4*, 1–11.

Schwindt, C. (2015). *Handbook on project management and scheduling* (Vol. 1). Springer.

Schwindt, C., & Trautmann, N. (2000). Batch scheduling in process industries: An application of resource-constrained project scheduling. *Or Spektrum*, *22*(4), 501–524.

Sprecher, A. (2002). Network decomposition techniques for resource-constrained project scheduling. *Journal of the Operational Research Society*, *53*(4), 405–414.

Stiti, C., & Driss, O. B. (2019). A new approach for the multi-site resource-constrained project scheduling problem. *Procedia Computer Science*, *164*, 478–484.

Thomas, P. R., & Salhi, S. (1998). A tabu search approach for the resource constrained project scheduling problem. *Journal of heuristics*, *4*, 123–139.

Tirkolaee, E. B., Goli, A., Faridnia, A., Soltani, M., & Weber, G.-W. (2020). Multi-objective optimization for the reliable pollution-routing problem with cross-dock selection using pareto-based algorithms. *Journal of Cleaner Production*, *276*, 122927.

Tormos, P., & Lova, A. (2001). Tools for resource-constrained project scheduling and control: Forward and backward slack analysis. *The Journal of the Operational Research Society*, *52*(7), 779–788.

Tormos, P., & Lova, A. (2003). An efficient multi-pass heuristic for project scheduling with constrained resources. *International journal of production research*, *41*(5), 1071–1086.

Trautmann, N., Fink, R., Sagebiel, H., & Schwindt, C. (2008). A decomposition approach to short-term scheduling of multi-purpose batch processes. In *Computer aided chemical engineering* (Vol. 25, pp. 157–162). Elsevier.

Trautmann, N., & Schwindt, C. (2005). A minlp/rcpsp decomposition approach for the short-term planning of batch production. In *computer aided chemical engineering* (Vol. 20, pp. 1309–1314). Elsevier.

Xie, L., Chen, Y., & Chang, R. (2021). Scheduling optimization of prefabricated construction projects by genetic algorithm. *Applied Sciences*, *11*(12), 5531.

Yang, D. Y., & Frangopol, D. M. (2020). Life-cycle management of deteriorating bridge networks with network-level risk bounds and system reliability analysis. *Structural safety*, *83*, 101911.

Yee, K. L. (1998). Efficient algorithms for multipurpose plant scheduling. *University of London PhD Thesis*.

Yin, Y. (2022). *Modelling and solving the multi-calendar naval surface ship work period problem under activitiy relationship constraints* (Unpublished master's thesis). Dalhousie University.

Zaman, F., Elsayed, S., Sarker, R., & Essam, D. (2020). Hybrid evolutionary algorithm for large-scale project scheduling problems. *Computers & Industrial Engineering*, *146*, 106567.

Zhang, T., Banitalebi-Dehkordi, A., & Zhang, Y. (2022). Deep reinforcement learning for exact combinatorial optimization: Learning to branch. In *2022 26th international conference on pattern recognition (ICPR)* (pp. 3105–3111).

Appendix A

Information of Computer Used for Experiments

Processor:10th Generation Intel® Core™ i7-10510U Processor (1.80 GHz, up to 4.90 GHz with Turbo Boost, 4 Cores, 8 Threads, 8 MB Cache)
Operating system:Windows 10 Pro
Memory:16 GB DDR4 2667MHz
Storage:1 TB PCIe SSD

Appendix B

Detailed Experiment Data

Table B.1: Comparison Total Time with 136 activities

Total time	136/2	136/3	136/4	136/5	136/6
SGS	0:00:34.7	0:00:32.8	0:00:30.7	0:00:37.2	0:00:35.3
BIP (5%)	0:06:16.1	0:06:13.1	0:06:20.2	0:06:14.9	0:06:25.4
BIP (1%)	0:06:15.0	0:06:16.0	0:06:15.8	0:06:14.2	0:06:18.1
BIP (0.5%)	0:06:14.7	0:06:14.6	0:06:16.8	0:06:18.9	0:06:22.1
BIP (0.1%)	0:06:17.3	0:06:20.0	0:06:21.0	0:06:15.4	0:06:17.5
MSO–4 ($\beta = 12$)	0:08:12.1	0:08:09.3	0:08:09.4	0:08:12.5	0:08:13.1
MSO–4 ($\beta = 11$)	0:08:11.0	0:08:09.3	0:08:09.5	0:08:17.4	0:08:13.0
MSO–4 ($\beta = 10$)	0:08:19.4	0:08:09.3	0:08:12.4	0:08:13.7	0:08:12.9
MSO-4 ($\beta = 9$)	0:08:12.1	0:08:04.9	0:08:12.6	0:08:25.2	0:08:13.3
MSO-4 ($\beta = 8$)	0:08:13.5	0:08:03.5	0:08:04.8	0:08:05.3	0:08:07.3
MSO-5 ($\beta = 12$)	0:08:11.9	0:08:05.4	0:08:08.4	0:08:12.2	0:08:13.4
MSO-5 ($\beta = 11$)	0:08:09.5	0:08:07.1	0:08:08.4	0:08:10.7	0:08:08.8
MSO-5 ($\beta = 10$)	0:08:09.1	0:08:07.3	0:08:08.5	0:08:05.1	0:08:07.6
MSO-5 ($\beta = 9$)	0:08:10.9	0:08:08.7	0:08:04.4	0:08:13.7	0:08:06.9
MSO-5 ($\beta = 8$)	0:08:14.1	0:08:12.4	0:08:10.1	0:08:10.3	0:08:09.3

Table B.2: Comparison Total Time with 680 activities

Total Time	680/2	680/3	680/4	680/5	680/6
SGS	0:11:33.5	0:11:39.7	0:11:34.1	0:11:33.3	0:11:31.1
BIP (5%)	2:10:57.9	2:14:16.1	2:12:10.3	2:11:51.1	2:13:53.1
BIP (1%)	2:18:07.3	3:29:33.1	2:26:21.8	2:25:48.0	2:48:06.8
BIP (0.5%)	2:38:06.3	4:04:40.4	2:40:31.2	2:36:13.1	3:19:58.4
BIP (0.1%)	4:04:07.2	4:04:41.9	3:39:25.0	3:25:05.3	3:40:40.4
MSO-4 ($\beta = 12$)	2:41:20.5	2:47:29.2	2:45:41.5	2:43:38.2	2:43:29.4
MSO-4 ($\beta = 11$)	2:45:42.0	2:46:27.8	2:44:12.9	2:40:58.5	2:42:06.3
MSO-4 ($\beta = 10$)	2:42:03.2	2:45:49.2	2:43:57.4	2:44:15.3	2:44:18.9
MSO-4 ($\beta = 9$)	2:43:05.5	2:44:22.8	2:44:35.4	2:43:27.6	2:43:24.6
MSO-4 ($\beta = 8$)	2:42:32.3	2:45:47.8	2:42:09.7	2:41:58.7	2:44:53.0
MSO-5 ($\beta = 12$)	2:40:56.2	2:45:19.1	2:42:15.1	2:43:55.4	2:44:48.5
MSO-5 ($\beta = 11$)	2:44:26.5	2:45:34.3	2:44:24.4	2:41:54.9	2:42:00.7
MSO-5 ($\beta = 10$)	2:43:28.5	2:45:49.2	2:42:47.6	2:43:19.8	2:42:58.8
MSO-5 ($\beta = 9$)	2:42:16.3	2:43:57.7	2:43:51.0	2:44:21.1	2:41:04.5
MSO-5 ($\beta = 8$)	2:41:20.2	2:43:50.5	2:43:54.9	2:43:34.0	2:41:31.6

Table B.3: Comparison Objective with 136 activities

Z- value	136/2	136/3	136/4	136/5	136/6
SGS	2757.3	1715.9	1573.6	3022.3	3022.3
BIP (5%)	2242.7	1694.9	1507.9	2882.8	2172.8
BIP (1%)	2195.7	1642.6	1475.6	2851.0	2144.0
BIP (0.5%)	2195.7	1642.6	1475.6	2840.8	2135.3
BIP (0.1%)	2193.5	1639.2	1471.8	2835.1	2135.3
MSO-4 ($\beta = 12$)	2192.6	1635.2	1466.8	2835.3	2139.5
MSO-4 ($\beta = 11$)	2190.7	1635.2	1466.9	2835.3	2140.0
MSO-4 ($\beta = 10$)	2190.7	1639.2	1466.9	2835.3	2140.0
MSO-4 ($\beta = 9$)	2190.8	1638.9	1466.3	2835.3	2140.6
MSO-4 ($\beta = 8$)	2190.7	1635.1	1461.5	2829.6	2140.6
MSO-5 ($\beta = 12$)	2676.7	2058.3	2181.4	4933.9	3319.9
MSO-5 ($\beta = 11$)	2676.7	2058.3	2181.4	4934.0	3319.9
MSO-5 ($\beta = 10$)	2676.7	2058.3	2181.4	4934.2	3319.9
MSO-5 ($\beta = 9$)	2676.7	2058.3	2220.4	4934.3	3319.9
MSO-5 ($\beta = 8$)	2674.3	2058.3	2203.7	4933.9	3319.9

Table B.4: Comparison Objective with 680 activities

Z- value	680/2	680/3	680/4	680/5	680/6
SGS	11537.3	14892.6	18780.8	18428.4	16834.2
BIP (5%)	8417.1	10113.2	13373.3	13986.3	12754.5
BIP (1%)	8256.6	9887.7	13135.7	13657.1	12516.9
BIP (0.5%)	8227.7	9881.8	13116.9	13631.3	12450.9
BIP (0.1%)	8221.7	9881.8	13079.3	13629.2	12449.8
MSO-4 ($\beta = 12$)	8808.5	9872.8	13872.1	14612.8	12720.6
MSO-4 ($\beta = 11$)	8807.4	9871.8	13889.6	14608.7	12722.5
MSO-4 ($\beta = 10$)	8698.4	9867.7	13879.4	14603.5	12722.9
MSO-4 ($\beta = 9$)	8702.7	9852.4	14122.4	14665.7	12718.5
MSO-4 ($\beta = 8$)	8807.4	9846.5	13904.8	14643.0	12766.8
MSO-5 ($\beta = 12$)	49958.6	64346.8	67129.2	67954.2	70659.0
MSO-5 ($\beta = 11$)	49818.6	64375.3	66670.8	68323.7	70818.7
MSO-5 ($\beta = 10$)	50022.5	64347.6	66582.6	68357.8	70350.5
MSO-5 ($\beta = 9$)	50072.8	64283.3	67814.6	68078.1	70476.0
MSO-5 ($\beta = 8$)	50927.9	64254.8	67754.7	67913.4	70471.9

Table B.5: Comparison optimality gap with 136 activities

Gap %	136/2	136/3	136/4	136/5	136/6
SGS	27.002%	6.538%	8.080%	9.365%	44.729%
BIP (5%)	3.194%	4.971%	3.447%	4.141%	3.895%
BIP (1%)	0.244%	0.373%	0.378%	0.741%	0.631%
BIP (0.5%)	0.244%	0.373%	0.378%	0.325%	0.191%
BIP (0.1%)	0.081%	0.072%	0.078%	0.056%	0.090%
MSO-4 ($\beta = 12$)	0.993%	1.525%	0.747%	2.600%	2.458%
MSO-4 ($\beta = 11$)	0.903%	1.525%	0.755%	2.600%	2.481%
MSO-4 ($\beta = 10$)	0.903%	1.773%	0.754%	2.599%	2.481%
MSO-4 ($\beta = 9$)	0.908%	1.756%	0.715%	2.599%	2.510%
MSO-4 ($\beta = 8$)	0.901%	1.523%	0.383%	2.394%	2.510%
MSO-5 ($\beta = 12$)	23.288%	27.794%	49.832%	78.540%	58.981%
MSO-5 ($\beta = 11$)	23.288%	27.794%	49.832%	78.544%	58.981%
MSO-5 ($\beta = 10$)	23.288%	27.794%	49.832%	78.553%	58.981%
MSO-5 ($\beta = 9$)	23.288%	27.794%	52.505%	78.557%	58.981%
MSO-5 ($\beta = 8$)	23.179%	27.794%	51.363%	78.541%	58.981%

Table B.6: Comparison optimality gap with 680 activities

Gap %	680/2	680/3	680/4	680/5	680/6
SGS	42.189%	54.915%	46.686%	38.336%	38.428%
BIP (5%)	3.603%	4.942%	4.262%	4.753%	4.654%
BIP (1%)	0.909%	0.983%	0.978%	0.976%	0.969%
BIP (0.5%)	0.442%	0.921%	0.453%	0.442%	0.444%
BIP (0.1%)	0.097%	0.921%	0.096%	0.100%	0.082%
MSO-4 ($\beta = 12$)	8.561%	2.699%	8.347%	9.693%	4.602%
MSO-4 ($\beta = 11$)	8.548%	2.688%	8.483%	9.663%	4.618%
MSO-4 ($\beta = 10$)	7.204%	2.645%	8.404%	9.624%	4.622%
MSO-4 ($\beta = 9$)	7.258%	2.486%	10.302%	10.091%	4.586%
MSO-4 ($\beta = 8$)	8.548%	2.425%	8.603%	9.920%	4.982%
MSO-5 ($\beta = 12$)	515.720%	569.346%	424.308%	410.110%	481.034%
MSO-5 ($\beta = 11$)	513.994%	569.643%	420.727%	412.884%	482.348%
MSO-5 ($\beta = 10$)	516.506%	569.354%	420.039%	413.140%	478.498%
MSO-5 ($\beta = 9$)	517.126%	568.685%	429.661%	411.041%	479.530%
MSO-5 ($\beta = 8$)	527.666%	568.389%	429.193%	409.804%	479.496%

Table B.7: Comparison DWC with 680 activities

DWC	136/2	136/3	136/4	136/5	136/6
SGS	358.6	340.4	308.8	311.1	321.2
BIP (5%)	365.2	357.7	277.7	292.7	268.0
BIP (1%)	284.8	302.5	261.9	274.9	253.3
BIP (0.5%)	284.8	302.5	261.9	254.8	255.9
BIP (0.1%)	283.4	291.1	254.7	252.6	255.9
MSO-4 ($\beta = 12$)	284.0	280.7	245.1	248.2	256.3
MSO-4 ($\beta = 11$)	279.5	280.7	244.8	248.2	257.2
MSO-4 ($\beta = 10$)	279.5	290.3	244.3	248.2	257.2
MSO-4 ($\beta = 9$)	278.5	289.7	244.5	248.2	259.0
MSO-4 ($\beta = 8$)	284.5	280.5	244.5	248.0	259.0
MSO-5 ($\beta = 12$)	280.8	245.3	209.9	296.5	269.3
MSO-5 ($\beta = 11$)	280.9	244.9	209.9	296.1	269.3
MSO-5 ($\beta = 10$)	280.8	245.3	209.9	296.7	269.3
MSO-5 ($\beta = 9$)	280.8	245.3	209.9	296.7	269.3
MSO-5 ($\beta = 8$)	280.2	245.3	208.7	296.4	269.3

Table B.8: Comparison DWC with 680 activities

DWC	680/2	680/3	680/4	680/5	680/6
SGS	1367.9	1429.1	1462.2	1441.5	1401.0
BIP (5%)	1255.3	1312.5	1270.0	1307.0	1242.2
BIP (1%)	1242.8	1305.4	1258.8	1279.7	1239.1
BIP (0.5%)	1238.8	1303.4	1256.8	1278.3	1234.0
BIP (0.1%)	1236.0	1303.4	1257.8	1278.3	1233.4
MSO-4 ($\beta = 12$)	1130.8	1179.2	1161.9	1119.8	1073.0
MSO-4 ($\beta = 11$)	1140.7	1176.4	1169.6	1120.8	1074.2
MSO-4 ($\beta = 10$)	1147.9	1175.1	1163.6	1118.8	1073.4
MSO-4 ($\beta = 9$)	1147.9	1175.3	1163.3	1120.3	1072.9
MSO-4 ($\beta = 8$)	1133.0	1178.9	1166.5	1122.8	1074.2
MSO-5 ($\beta = 12$)	1003.7	1294.7	1222.6	1273.9	1305.5
MSO-5 ($\beta = 11$)	1007.0	1294.4	1215.6	1275.1	1308.1
MSO-5 ($\beta = 10$)	1005.1	1294.6	1211.3	1276.1	1301.8
MSO-5 ($\beta = 9$)	1003.5	1293.0	1231.2	1274.4	1304.7
MSO-5 ($\beta = 8$)	1012.8	1290.4	1230.7	1273.4	1303.6

Table B.9: Comparison CPU time with 136 activities

CPU time(s)	136/2	136/3	136/4	136/5	136/6
SGS					
BIP (5%)	2.03	2.31	3.24	1.89	2.01
BIP (1%)	3.67	3.84	4.05	2.22	3.97
BIP (0.5%)	3.67	3.93	4.1	3.53	4.01
BIP (0.1%)	5.20	5.00	5.39	4.19	4.09
MSO-4 ($\beta = 12$)	2.69	2.26	2.09	2.28	2.11
MSO-4 ($\beta = 11$)	2.70	2.26	2.09	2.30	2.18
MSO-4 ($\beta = 10$)	2.68	2.23	2.10	2.33	2.11
MSO-4 ($\beta = 9$)	2.82	2.18	2.16	2.37	2.22
MSO-4 ($\beta = 8$)	2.56	2.32	2.17	2.55	2.29
MSO-5 ($\beta = 12$)	2.64	4.42	2.81	2.88	2.80
MSO-5 ($\beta = 11$)	2.57	4.13	3.20	2.94	2.55
MSO-5 ($\beta = 10$)	2.58	4.30	2.87	2.80	2.81
MSO-5 ($\beta = 9$)	2.60	4.22	2.91	2.91	2.53
MSO-5 ($\beta = 8$)	2.67	4.77	2.63	2.88	2.68

Table B.10: Comparison CPU time with 680 activities

CPU time(s)	680/2	680/3	680/4	680/5	680/6
SGS					
BIP (5%)	196.07	267.80	240.33	239.77	411.26
BIP (1%)	493.58	4849.24	982.69	1039.43	2420.34
BIP (0.5%)	1840.21	6902.66	1835.92	1692.44	4322.90
BIP (0.1%)	6905.54	6902.88	5394.41	4625.73	5592.53
MSO-4 ($\beta = 12$)	33.81	42.01	41.44	40.05	36.44
MSO-4 ($\beta = 11$)	35.27	43.11	40.07	40.60	36.37
MSO-4 ($\beta = 10$)	35.10	42.08	40.71	39.74	39.08
MSO-4 ($\beta = 9$)	33.10	42.76	39.96	42.08	39.12
MSO-4 ($\beta = 8$)	33.21	42.14	40.49	39.94	39.56
MSO-5 ($\beta = 12$)	54.13	65.69	59.86	71.98	58.00
MSO-5 ($\beta = 11$)	60.00	71.34	60.47	74.26	60.48
MSO-5 ($\beta = 10$)	56.82	65.33	62.66	75.32	60.35
MSO-5 ($\beta = 9$)	56.08	70.31	61.87	75.14	56.94
MSO-5 ($\beta = 8$)	54.33	64.25	62.57	71.58	61.64

Table B.11: Comparison subgroup sizes with MSO-4 and 680 activities

Subgroup Size	Z value	DWC	Makespan	CPU time(s)	Total time
2	14854.0	1110.9	588	14.1	2:44:32.2
3	14751.7	1103.9	595	12.9	2:45:09.5
5	14651.0	1110.8	630	15.7	2:43:30.2
10	14648.2	1119.6	630	25.3	2:45:51.8
20	14620.2	1119.7	623	44.9	2:44:50.5
30	14608.7	1120.8	630	40.6	2:40:58.5
50	14626.4	1120.1	630	48.8	2:45:25.3
70	13923.7	1123.9	630	97.4	2:43:34.7
100	14388.2	1110.2	616	130.2	2:44:53.8

Table B.12: Comparison subgroup sizes with MSO-5 and 680 activities

Subgroup Size	Z value	DWC	Makespan	CPU time(s)	Total time
30	68323.7	1275.1	609	74.3	2:41:54.9
50	67669.1	1302.7	609	85.0	2:42:46.0
70	67456.8	1315.3	629	260.2	2:44:38.4
100	62959.1	1303.2	623	1736.4	3:09:19.0

Appendix C

Python Code of the BIP model

```
# Create the problem called "problem1"
prob = LpProblem("problem1", LpMinimize)

# Create the variable x[j,t,m]
var =
    [[[pulp.LpVariable("x[%s,%s,%s" % (j + 1, t
 + ES.iloc[j], m + 1), None,
     None, LpBinary) for m in range(M.shape[0])]
      for t in
      range(LS.iloc[j] - ES.iloc[j] + 1)]
     for j in range(n)]

# Set the initial value
for j in range(len(var)):
    for t in range(len(var[j])):
        for m in range(len(var[j][t])):
            var[j][t][m].setInitialValue(0)

for i in range(asgs):
    var[i][SGS['ES'].iat[i] - ES.iloc[i]][0].setInitialValue(1)



# Create the objective function


prob += pulp.lpSum(
    [[[var[j][t][m] * (t + ES.iloc[j] + 1) * (1 / (PR.iloc[j]
 ** 5)) * \

      ((0.001 + int(D.iloc[j, t + ES.iloc[j]][m])) ** 1.1)
*((m+1)**11)
    for m in range(M.shape[0])]
    for t in range(LS.iloc[j] - ES.iloc[j] + 1)] for j in range(n)])

# Create constraint 1: all activities should be completed no more
 than 1 time between ES and LS
for j in range(n):
    prob += (pulp.lpSum([[var[j][t][m] for m in range
(M.shape[0])]
    for t in range(LS.iloc[j] - ES.iloc[j] + 1)]) == 1)

# Create constraint 2: finish to start
for i in range(C7.shape[0]):
    # print(P["element id"].iat[i])
    j1 = int(C7["pred"].iat[i]) - 1
    j2 = int(C7["succ"].iat[i]) - 1
    lag = int(C7["lag"].iat[i])
    prob += (pulp.lpSum([[(t + ES.iat[j2]) *
 var[j2][t][m]
    for t in range(LS.iat[j2] - ES.iat[j2] + 1)]
    for m in range(M.shape[0])]) >=
             pulp.lpSum([[var[j1][t][m] * ((t + ES.iat[j1]) +
             D.iat[j1, t + ES.iloc[j1]][m] + lag) for t in
                          range(LS.iat[j1] - ES.iat[j1] + 1)]
                         for m in range(M.shape[0])]))

# Create constraint 3: finish to finish
```

```
for i in range(C8.shape[0]):
    # print(P["element id"].iat[i])
    j1 = int(C8["pred"].iat[i]) - 1
    j2 = int(C8["succ"].iat[i]) - 1
    lag = int(C8["lag"].iat[i])
    prob += (pulp.lpSum(
        [[(t + ES.iat[j2] + D.iat[j2, t + ES.iloc[j2]][m]) *
 var[j2][t][m]
          for t in range(LS.iat[j2] - ES.iat[j2] + 1)]
          for m in range(M.shape[0])]) >=
             pulp.lpSum([[var[j1][t][m] * ((t + ES.iat[j1]) +
              D.iat[j1, t + ES.iloc[j1]][m] + lag - 1) \
                          for t in range(LS.iat[j1] - ES.iat[j1] + 1)]
                         for m in range(M.shape[0])]))

# Create constraint 4: Start to start
for i in range(C9.shape[0]):
    # print(P["element id"].iat[i])
    j1 = int(C9["pred"].iat[i]) - 1
    j2 = int(C9["succ"].iat[i]) - 1
    lag = int(C9["lag"].iat[i])
    prob += (pulp.lpSum(
        [[(t + ES.iat[j2]) * var[j2][t][m]
        for t in range(LS.iat[j2] - ES.iat[j2] + 1)]
        for m in range(M.shape[0])]) >=
             pulp.lpSum([[var[j1][t][m] * (t + ES.iat[j1] + lag)
             for t in range(LS.iat[j1] - ES.iat[j1] + 1)]
                         for m in range(M.shape[0])]))

# Create constraint 5: Start to finish
for i in range(C10.shape[0]):
    # print(P["element id"].iat[i])
    j1 = int(C10["pred"].iat[i]) - 1
    j2 = int(C10["succ"].iat[i]) - 1
    lag = int(C10["lag"].iat[i])
    prob += (pulp.lpSum(
        [[(t + ES.iat[j2] + D.iat[j2, t +
         ES.iloc[j2]][m] - 1) * var[j2][t][m]
          for t in range(LS.iat[j2] - ES.iat[j2] + 1)]
          for m in range(M.shape[0])]) >=
             pulp.lpSum([[var[j1][t][m] * (t + ES.iat[j1] + lag)
             for t in range(LS.iat[j1] - ES.iat[j1] + 1)]
                         for m in range(M.shape[0])]))

# Create constraint 6: finish on
for i in range(C2.shape[0]):
    # print(P["element id"].iat[i])
    j1 = int(C2.iloc[i, 0]) - 1
    j2 = int(C2.iloc[i, 1])
    prob += (pulp.lpSum(
        [[var[j1][t][m] * (t + ES.iat[j1] + D.iat[j1, t +
         ES.iloc[j1]][m] - 1) for m in range(M.shape[0])] \
         for t in range(LS.iloc[j1] - ES.iloc[j1] + 1)]) == j2)

# Create constraint 7: finish on or after
for i in range(C4.shape[0]):
    # print(P["element id"].iat[i])
    j1 = int(C4.iloc[i, 0]) - 1
    j2 = int(C4.iloc[i, 1])
    prob += (pulp.lpSum([[var[j1][t][m] * (t + ES.iat[j1]
    + D.iat[j1, t + ES.iloc[j1]][m] - 1) \
                          for m in range(M.shape[0])]
                          for t in range(LS.iloc[j1] -
                           ES.iloc[j1] + 1)]) >= j2)
```

```
# Create constraint 8: finish on or before
for i in range(C6.shape[0]):
    j1 = int(C6.iloc[i, 0]) - 1
    j2 = int(C6.iloc[i, 1])
    prob += (pulp.lpSum([[var[j1][t][m] * (t + ES.iat[j1]
     + D.iat[j1, t + ES.iloc[j1]][m] - 1)
          for m in range(M.shape[0])]
          for t in range(LS.iloc[j1] - ES.iloc[j1] + 1)]) <= j2)

# Create constraint 9: resource availability
for k in range(int(R)):
    for s in range(int(H + 1)):
        temp = []
        for j in range(n):
            for m in range(M.shape[0]):
                for t in range(int(max(s - DB.iloc[j, s][m] + 1,
 ES[j])
                  - ES.iloc[j]), int(min(LS[j], s) - ES.iloc[j] + 1)):
                    form1 = RD.iloc[j, k] * var[j][t][m] *
                    AC.iloc[int(SGS["calendar"].iat[j]), s]
                    temp.append(form1)
        prob += (pulp.lpSum(temp) <= AV.iat[k, s])

# Create the .lp file
prob.writeLP("problem1.lp")

prob.solve(GUROBI_CMD(mip=MIP,options=[
("Heuristics", 0.2),
  ("Symmetry", 2)],
   warmStart=True,
   gapRel=Tol))
```

Appendix D

Python Code of the MSO-4 model

```
for B in range((asgs-1)//50 + 1):
    #create iteration
    BlistC += list(batch(Blist,50))[B]

    SGS.sort_values("Activity", inplace=True)
    SGS = SGS.reset_index(drop=True)
    #create Mode(m)
    M = []

    for i in range(q):

        M.append(i)

    for i in range(previous_len,len(BlistC)):

        ModeA.append([M])

    Modes = pd.DataFrame()
    Modes['Mode'] = ModeA

    # Create the problem called "problem1"
    prob = LpProblem("problem1", LpMinimize)

    # Create the variable x[j,t,m]
    var = \
        [[[pulp.LpVariable("x[%s,%s,%s" % (BlistC[j] + 1,
 t + ES.ia

t[BlistC[j]], m + 1),
                           lowBound=None, upBound=None,
cat=const.
LpBinary, e=None)
           for m in ModeA[j][0]] for t in range(LS.iat[
BlistC[j]]
 - ES.iat[BlistC[j]] + 1)] for j in range(len(BlistC))]

    # Set the initial value
    for j in range(len(BlistC)):
        for t in range(len(var[j])):
            for m in range(len(var[j][t])):
                var[j][t][m].setInitialValue(0)

    for i in range(len(BlistC)):
        if SGS['ES'].iat[BlistC[i]] - ES.iat[BlistC[i]] >= 0:
            var[i][SGS['ES'].iat[BlistC[i]]-ES.iat[BlistC[i]]][0]
.setInitialValue(1)
        elif SGS['ES'].iat[BlistC[i]] - ES.iat[BlistC[i]] <= 0:
            var[i][0][0].setInitialValue(1)

    #Objective(Minimize)
    temp_arr = []

    for j in range(len(BlistC)):
        for t in range(LS.iat[BlistC[j]] - ES.iat[BlistC[j]] + 1):
            if ModeA[j][0] ==[0,1,2,3]:
                for m in ModeA[j][0]:
                    form = var[j][t][m] * (t +
ES.iloc[BlistC[j]] + 1) * (1 / (PR.iloc[BlistC[j]] ** 5)) * \
                           ((0.001 + int(D.iloc[BlistC[j],
 t + ES.iloc[BlistC[j]]][m])) ** 1.1) * ((m + 1) ** 10)
                    temp_arr.append(form)
            elif ModeA[j][0] != [0, 1, 2, 3]:
                for m in ModeA[j][0]:
                    form = var[j][t][0] * (t +
ES.iloc[BlistC[j]] + 1) *
 (1 / (PR.iloc[BlistC[j]] ** 5)) * \
                           ((0.001 + int(D.iloc[BlistC[j],
t + ES.iloc[BlistC[j]]][m])) ** 1.1) * ((m + 1) ** 8)

    prob += pulp.lpSum(temp_arr)

    # Creat constraint 0: All activities cannot start on holiday

    for j in range(previous_len, len(BlistC)):
        for t in range(LS.iloc[BlistC[j]] -
 ES.iloc[BlistC[j]] + 1):
            prob += (pulp.lpSum([var[j][t][m] for
m in ModeA[j][0]]) \
                    <= AC.iat[int(SGS["calendar"]
.iat[BlistC[j]]),
t + ES.iat[BlistC[j]]]])

    # Create constraint 1: all activities should be completed no
 more than 1 time between ES and LSint(C8["pred"].iat[i] - 1) in Bl

    for j in range(len(BlistC)):
        temp_arr = []
        for t in range(LS.iat[BlistC[j]]
- ES.iat[BlistC[j]] + 1):
            if ModeA[j][0] == [0, 1, 2, 3]:
                for m in ModeA[j][0]:
                    form = var[j][t][m]
                    temp_arr.append(form)

            elif ModeA[j][0] != [0, 1, 2, 3]:
                for m in ModeA[j][0]:
                    form = var[j][t][0]
                    temp_arr.append(form)

        prob += pulp.lpSum(temp_arr) == 1

    # Create constraint 2: finish to start

    for i in range(C7.shape[0]):

        lag = int(C7["lag"].iat[i])

        if int(C7["pred"].iat[i]-1) in BlistC and int
(C7["succ"].iat[i]
-1) in BlistC:
            j1 = BlistC.index(int(C7["pred"].iat[i]-1))
            j2 = BlistC.index(int(C7["succ"].iat[i]-1))
            j3 = int(C7["pred"].iat[i]) - 1
            j4 = int(C7["succ"].iat[i]) - 1
            temp_arry1 = []
            temp_arry2 = []

            for t in range(LS.iat[j4] - ES.iat[j4] + 1):

                if ModeA[j2][0] == [0, 1, 2, 3]:
                    for m in ModeA[j2][0]:
                        form1 = (t + ES.iat[j4]) * var[j2][t][m]
                        temp_arry1.append(form1)

                elif ModeA[j2][0] != [0,1,2,3]:
                    for m in ModeA[j2][0]:
                        form1 = (t + ES.iat[j4]) * var[j2][t][0]
                        temp_arry1.append(form1)

            for t in range(LS.iat[j3] - ES.iat[j3] + 1):
                if ModeA[j1][0] == [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form2 = var[j1][t][m] * ((t + ES.iat[j3])
+ D.iat[j3, t + ES.iloc[j3]][m] + lag)
                        temp_arry2.append(form2)
                elif ModeA[j1][0] != [0,1,2,3]:
                    for m in ModeA[j1][0]:
                        form2 = var[j1][t][0] * ((t + ES.iat[j3])
+ D.iat[j3, t + ES.iloc[j3]][m] + lag)
                        temp_arry2.append(form2)

            prob += (pulp.lpSum(temp_arry1)
>= pulp.lpSum(temp_arry2))

    # Create constraint 3: finish to finish

    for i in range(C8.shape[0]):
        #print(P["element id"].iat[i])
        lag = int(C8["lag"].iat[i])
        if int(C8["pred"].iat[i] - 1) in BlistC
and int(C8["succ"].iat[i] - 1) in BlistC:
            j1 = BlistC.index(int(C8["pred"].iat[i] - 1))
            j2 = BlistC.index(int(C8["succ"].iat[i] - 1))
            j3 = int(C8["pred"].iat[i]) - 1
            j4 = int(C8["succ"].iat[i]) - 1
            temp_arry1 = []
            temp_arry2 = []
            for t in range(LS.iat[j4] - ES.iat[j4] + 1):
                if ModeA[j2][0] == [0, 1, 2, 3]:
                    for m in ModeA[j2][0]:
                        form1 = (t + ES.iat[j4] + D.iat[j4,
 t + ES.iloc[j4]][m]) * var[j2][t][m]
                        temp_arry1.append(form1)

                elif ModeA[j2][0] != [0,1,2,3]:
                    for m in ModeA[j2][0]:
                        form1 = (t + ES.iat[j4] + D.iat[j4,
 t + ES.iloc[j4]][m]) * var[j2][t][0]
                        temp_arry1.append(form1)

            for t in range(LS.iat[j3] - ES.iat[j3] + 1):
                if ModeA[j1][0] == [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form2 = var[j1][t][m] * ((t + ES.iat[j3])
+ D.iat[j3, t + ES.iloc[j3]][m] + lag - 1)
                        temp_arry2.append(form2)
                elif ModeA[j1][0] != [0,1,2,3]:
                    for m in ModeA[j1][0]:
                        form2 = var[j1][t][0] * ((t + ES.iat[j3])
+ D.iat[j3, t + ES.iloc[j3]][m] + lag - 1)
                        temp_arry2.append(form2)

            prob += (pulp.lpSum(temp_arry1)
>= pulp.lpSum(temp_arry2))
```

```
    # Create constraint 4: Start to start

    for i in range(C9.shape[0]):
        #print(P["element id"].iat[i])
        lag = int(C9["lag"].iat[i])
        if int(C9["pred"].iat[i] - 1) in BlistC and int(
C9["succ"].iat[i] - 1) in BlistC:
            j1 = BlistC.index(int(C9["pred"].iat[i] - 1))
            j2 = BlistC.index(int(C9["succ"].iat[i] - 1))
            j3 = int(C9["pred"].iat[i]) - 1
            j4 = int(C9["succ"].iat[i]) - 1
            temp_arry1 = []
            temp_arry2 = []
            for t in range(LS.iat[j4] - ES.iat[j4] + 1):
                if ModeA[j2][0] == [0, 1, 2, 3]:
                    for m in ModeA[j2][0]:
                        form1 = (t + ES.iat[j4]) * var[j2][t][m]
                        temp_arry1.append(form1)

                elif ModeA[j2][0] != [0,1,2,3]:
                    for m in ModeA[j2][0]:
                        form1 = (t + ES.iat[j4]) * var[j2][t][0]
                        temp_arry1.append(form1)

            for t in range(LS.iat[j3] - ES.iat[j3] + 1):
                if ModeA[j1][0] == [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form2 = var[j1][t][m] * (t + ES.iat[j3] + lag)
                        temp_arry2.append(form2)
                elif ModeA[j1][0] != [0,1,2,3]:
                    for m in ModeA[j1][0]:
                        form2 = var[j1][t][0] * (t + ES.iat[j3] + lag)
                        temp_arry2.append(form2)

            prob += pulp.lpSum(temp_arry1)
>= pulp.lpSum(temp_arry2)

    # Create constraint 5: Start to finish

    for i in range(C10.shape[0]):
        #print(P["element id"].iat[i])
        lag = int(C10["lag"].iat[i])
        if int(C10["pred"].iat[i] - 1) in BlistC and
int(C10["succ"].iat[i] - 1) in BlistC:
            j1 = BlistC.index(int(C10["pred"].iat[i] - 1))
            j2 = BlistC.index(int(C10["succ"].iat[i] - 1))
            j3 = int(C10["pred"].iat[i]) - 1
            j4 = int(C10["succ"].iat[i]) - 1
            temp_arry1 = []
            temp_arry2 = []
            for t in range(LS.iat[j4] - ES.iat[j4] + 1):
                if ModeA[j2][0] == [0, 1, 2, 3]:
                    for m in ModeA[j2][0]:
                        form1 = (t + ES.iat[j4] + D.iat[j4,
 t + ES.iloc[j4]][m] - 1) * var[j2][t][m]
                        temp_arry1.append(form1)

                elif ModeA[j2][0] != [0,1,2,3]:
                    for m in ModeA[j2][0]:
                        form1 = (t + ES.iat[j4] + D.iat[j4,
 t + ES.iloc[j4]][m] - 1) * var[j2][t][0]
                        temp_arry1.append(form1)
            for t in range(LS.iat[j3] - ES.iat[j3] + 1):
                if ModeA[j1][0] == [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form2 = var[j1][t][m] * (t
+ ES.iat[j3] + lag)
                        temp_arry2.append(form2)
                elif ModeA[j1][0] != [0,1,2,3]:
                    for m in ModeA[j1][0]:
                        form2 = var[j1][t][0] * (t
+ ES.iat[j3] + lag)
                        temp_arry2.append(form2)

            prob += (pulp.lpSum(temp_arry1)
>= pulp.lpSum(temp_arry2))



    # Create constraint 6: Start on

    for i in range(C1.shape[0]):
        #print(P["element id"].iat[i])
        j3 = int(C1.iloc[i,1])
        if int(C1.iloc[i,0]-1) in BlistC:
            j1 = BlistC.index(int(C1.iloc[i,0]-1))
            j2 = int(C1.iloc[i,0]-1)
            temp_arr = []
            for t in range(LS.iat[j2] - ES.iat[j2] + 1):
                if ModeA[j1][0] == [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form1 = var[j1][t][m] * (t + ES.iat[j2])
                        temp_arr.append(form1)

                elif ModeA[j1][0] != [0,1,2,3]:
                    for m in ModeA[j1][0]:
                        form1 = var[j1][t][0] * (t + ES.iat[j2])
                        temp_arr.append(form1)

            prob += (pulp.lpSum(temp_arr) == j3)

    # Create constraint 7: finish on



    for i in range(C2.shape[0]):
        #print(P["element id"].iat[i])
        j3 = int(C2.iloc[i, 1])
        if int(C2.iloc[i, 0] - 1) in BlistC:
            j1 = BlistC.index(int(C2.iloc[i, 0] - 1))
            j2 = int(C2.iloc[i, 0] - 1)
            temp_arr = []
            for t in range(LS.iat[j2] - ES.iat[j2] + 1):
                if ModeA[j1][0] == [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form1 = var[j1][t][m] * (t + ES.iat[j2]
+ D.iat[j2, t + ES.iloc[j2]][m] - 1)
                        temp_arr.append(form1)

                elif ModeA[j1][0] != [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form1 = var[j1][t][0] * (t + ES.iat[j2]
+ D.iat[j2, t + ES.iloc[j2]][m] - 1)
                        temp_arr.append(form1)

            prob += (pulp.lpSum(temp_arr) == j3)

    #Create constraint 8: Start on or after
```

```
for i in range(C3.shape[0]):
    #print(P["element id"].iat[i])
    j3 = int(C3.iloc[i, 1])
    if int(C3.iloc[i, 0] - 1) in BlistC:
        j1 = BlistC.index(int(C3.iloc[i, 0] - 1))
        j2 = int(C3.iloc[i, 0] - 1)
        temp_arr = []

        for t in range(LS.iat[j2] - ES.iat[j2] + 1):
            if ModeA[j1][0] == [0, 1, 2, 3]:
                for m in ModeA[j1][0]:
                    form1 = var[j1][t][m] * (t + ES.iat[j2])
                    temp_arr.append(form1)

            elif ModeA[j1][0] != [0, 1, 2, 3]:
                for m in ModeA[j1][0]:
                    form1 = var[j1][t][0] * (t + ES.iat[j2])
                    temp_arr.append(form1)

        prob += (pulp.lpSum(temp_arr) >= j3)

# Create constraint 9: finish on or after

for i in range(C4.shape[0]):
    #print(P["element id"].iat[i])
    j3 = int(C4.iloc[i, 1])
    if int(C4.iloc[i, 0] - 1) in BlistC:
        j1 = BlistC.index(int(C4.iloc[i, 0] - 1))
        j2 = int(C4.iloc[i, 0] - 1)
        temp_arr = []
        for t in range(LS.iat[j2] - ES.iat[j2] + 1):
                if ModeA[j1][0] == [0, 1, 2, 3]:

                    for m in ModeA[j1][0]:
                        form1 = var[j1][t][m] * (t + ES.iat[j2]
+ D.iat[j2, t + ES.iloc[j2]][m] - 1)
                        temp_arr.append(form1)

                elif ModeA[j1][0] != [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form1 = var[j1][t][0] * (t + ES.iat[j2]
+ D.iat[j2, t + ES.iloc[j2]][m] - 1)
                        temp_arr.append(form1)

            prob += (pulp.lpSum(temp_arr) >= j3)

    # Create constraint 10: Start on or before

    for i in range(C5.shape[0]):
        #print(P["element id"].iat[i])

        j3 = int(C5.iloc[i, 1])
        if int(C5.iloc[i, 0] - 1) in BlistC:
            j1 = BlistC.index(int(C5.iloc[i, 0] - 1))
            j2 = int(C5.iloc[i, 0] - 1)
            temp_arr = []
            for t in range(LS.iat[j2] - ES.iat[j2] + 1):
                if ModeA[j1][0] == [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form1 = var[j1][t][m] * (t + ES.iat[j2])
                        temp_arr.append(form1)

                elif ModeA[j1][0] != [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form1 = var[j1][t][0] * (t + ES.iat[j2])
                        temp_arr.append(form1)

            prob += (pulp.lpSum(temp_arr) <= j3)

    # Create constraint 11: finish on or before

    for i in range(C6.shape[0]):
        j3 = int(C6.iloc[i, 1])
        if int(C6.iloc[i, 0] - 1) in BlistC:
            j1 = BlistC.index(int(C6.iloc[i, 0] - 1))
            j2 = int(C6.iloc[i, 0] - 1)
            temp_arr = []
            for t in range(LS.iat[j2] - ES.iat[j2] + 1):
                if ModeA[j1][0] == [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form1 = var[j1][t][m] * (t + ES.iat[j2]
+ D.iat[j2, t + ES.iloc[j2]][m] - 1)
                        temp_arr.append(form1)

                elif ModeA[j1][0] != [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form1 = var[j1][t][0] * (t + ES.iat[j2]
+ D.iat[j2, t + ES.iloc[j2]][m] - 1)
                        temp_arr.append(form1)

            prob += (pulp.lpSum(temp_arr) <= j3)

    # #Create constraint 12: resource availability

    for k in range(int(R)):
        for s in range(int(H + 1)):
            temp_arr = []
            for j in range(previous_len, len(BlistC)):
                if ModeA[j][0] == [0, 1, 2, 3]:
                    for m in ModeA[j][0]:
                        for t in range(int(max(s -
DB.iloc[BlistC[j], s][m] - 1+ AC.iloc[
int(SGS["calendar"].
iat[BlistC[j]]), s - DB.iloc[BlistC[j], s][m]]
+ AC.iloc[int(
SGS["calendar"].iat[BlistC[j]]),
s - DB.iloc[BlistC[j], s][
                            m] - 1],
                                                ES[BlistC[j]])
- ES[BlistC[j]]),
                                        int(min(LS[BlistC[j]], s)
- ES[BlistC[j]] + 1)):
                            form1 = RD.iloc[BlistC[j], k]
 * var[j][t][m] * AC.iloc[int(SGS["calendar"].
iat[BlistC[j]]), s]
                            temp_arr.append(form1)

                elif ModeA[j][0] != [0, 1, 2, 3]:
                    for m in ModeA[j][0]:
                        for t in range(int(max(s
- DB.iloc[BlistC[j], s][m] - 1 + AC.iloc[
int(SGS["calendar"].iat[BlistC[j]]),
 s - DB.iloc[BlistC[j], s][m]]
+ AC.iloc[int(SGS["calendar"].
iat[BlistC[j]]), s - DB.iloc[BlistC[j], s][
                            m] - 1], ES[BlistC[j]]) \
                                            - ES[BlistC[j]]),
 int(min(LS[BlistC[j]], s) - ES[BlistC[j]] + 1)):
                            form1 = RD.iloc[BlistC[j], k]
* var[j][t][0] * AC.iloc[int(SGS["calendar"].
iat[BlistC[j]]), s]

                            temp_arr.append(form1)

            prob += (pulp.lpSum(temp_arr) <= AV.iat[k, s])

    # Create the .lp file
    prob.writeLP("problem1.lp")

    # Solve the .lp file
    prob.solve(GUROBI_CMD(mip=3, options=
[("TimeLimit",
1500), ("Heuristics", 0.2), ("Symmetry", 2)],
warmStart=True))
```

Appendix E

Python Code of the MSO-5 model

```
for B in Blist:
    # create iteration
    BlistC = B

    SGS.sort_values("Activity", inplace=True)
    SGS = SGS.reset_index(drop=True)
    # create Mode(m)
    M = []
    del ModeA[:]

    for i in range(q):
        M.append(i)

    for i in range(len(BlistC)):
        ModeA.append([M])

    Modes = pd.DataFrame()
    Modes['Mode'] = ModeA

    # Create the problem called "problem1"
    prob = LpProblem("problem1", LpMinimize)

    # Create the variable x[j,t,m]
    var = \
        [[[pulp.LpVariable("x[%s,%s,%s" %
(BlistC[j] + 1, t + ES.iat[BlistC
[j]], m + 1), None, None, LpBinary)

           for m in ModeA[j][0]] for t in range
(LS.iat[BlistC[j]] - ES.iat[BlistC
[j]] + 1)] for j in range(len(BlistC))]

    # Set the initial value
    for j in range(len(BlistC)):
        for t in range(len(var[j])):
            for m in range(len(var[j][t])):
                var[j][t][m].setInitialValue(0)

    # Objective(Minimize)
    temp_arr = []

    for j in range(len(BlistC)):
        for t in range(LS.iat[BlistC[j]]
 - ES.iat[BlistC[j]] + 1):
            if ModeA[j][0] == [0, 1, 2, 3]:
                for m in ModeA[j][0]:
                    form = var[j][t][m] * (t
+ ES.iloc[BlistC[j]] + 1) * (1 / (PR.iloc
[BlistC[j]] ** 5)) * \
                           ((0.001 + int(D.iloc[BlistC[j],
 t + ES.iloc[BlistC[j]]][m]))
** 1.1) * ((m + 1) ** 11)
                    temp_arr.append(form)
            elif ModeA[j][0] != [0, 1, 2, 3]:
                for m in ModeA[j][0]:
                    form = var[j][t][0] * (t
+ ES.iloc[BlistC[j]] + 1) * (1 / (PR.iloc
[BlistC[j]] ** 5)) * \
                           ((0.001 + int(D.iloc[BlistC[j],
t + ES.iloc[BlistC[j]]][m]))
 ** 1.1) * ((m + 1) ** 11)
                    temp_arr.append(form)

    prob += pulp.lpSum(temp_arr)

    # Creat constraint 0: All activities cannot start on holiday

    for j in range(len(BlistC)):
        for t in range(LS.iloc[BlistC[j]]
 - ES.iloc[BlistC[j]] + 1):
 prob += (pulp.lpSum([var[j][t][m]
for m in ModeA[j][0]])
<= AC.iloc[int(SGS["calendar"].iat[BlistC[j]]),
t + ES.iloc[BlistC[j]]])

    # Create constraint 1: all activities should be completed no more
than 1 time between ES and LS

    for j in range(len(BlistC)):
        temp_arr = []
        for t in range(LS.iat[BlistC[j]]
- ES.iat[BlistC[j]] + 1):
            if ModeA[j][0] == [0, 1, 2, 3]:
                for m in ModeA[j][0]:
                    form = var[j][t][m]
                    temp_arr.append(form)
            elif ModeA[j][0] != [0, 1, 2, 3]:
                for m in ModeA[j][0]:
                    form = var[j][t][0]
                    temp_arr.append(form)
            else:
                form = var[j][t][0]
                temp_arr.append(form)

        prob += (pulp.lpSum(temp_arr) == 1)

    # Create constraint 2: finish to start

    for i in range(C7.shape[0]):

        lag = int(C7["lag"].iat[i])

        if int(C7["pred"].iat[i] - 1) in BlistC
and int(C7["succ"].iat[i] - 1) in BlistC:
            j1 = BlistC.index(int(C7["pred"].iat[i] - 1))
            j2 = BlistC.index(int(C7["succ"].iat[i] - 1))
            j3 = int(C7["pred"].iat[i]) - 1
            j4 = int(C7["succ"].iat[i]) - 1
            temp_arry1 = []
            temp_arry2 = []

            for t in range(LS.iat[j4] - ES.iat[j4] + 1):

                if ModeA[j2][0] == [0, 1, 2, 3]:
                    for m in ModeA[j2][0]:
                        form1 = (t + ES.iat[j4]) * var[j2][t][m]
                        temp_arry1.append(form1)

                elif ModeA[j2][0] != [0, 1, 2, 3]:
                    for m in ModeA[j2][0]:
                        form1 = (t + ES.iat[j4]) * var[j2][t][0]
                        temp_arry1.append(form1)

            for t in range(LS.iat[j3] - ES.iat[j3] + 1):
                if ModeA[j1][0] == [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form2 = var[j1][t][m] *
((t + ES.iat[j3]) + D.iat[j3, t
+ ES.iloc[j3]][m] + lag)
                        temp_arry2.append(form2)
                elif ModeA[j1][0] != [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form2 = var[j1][t][0] *
((t + ES.iat[j3]) + D.iat[j3, t
+ ES.iloc[j3]][m] + lag)
                        temp_arry2.append(form2)

            prob += (pulp.lpSum(temp_arry1)
>= pulp.lpSum(temp_arry2))

    # Create constraint 3: finish to finish

    for i in range(C8.shape[0]):
        # print(P["element id"].iat[i])
        lag = int(C8["lag"].iat[i])
        if int(C8["pred"].iat[i] - 1) in BlistC and
int(C8["succ"].iat[i] - 1) in BlistC:
            j1 = BlistC.index(int(C8["pred"].iat[i] - 1))
            j2 = BlistC.index(int(C8["succ"].iat[i] - 1))
            j3 = int(C8["pred"].iat[i]) - 1
            j4 = int(C8["succ"].iat[i]) - 1
            temp_arry1 = []
            temp_arry2 = []
            for t in range(LS.iat[j4] - ES.iat[j4] + 1):
                if ModeA[j2][0] == [0, 1, 2, 3]:
                    for m in ModeA[j2][0]:
                        form1 = (t + ES.iat[j4]
+ D.iat[j4, t + ES.iloc[j4]][m]) *
var[j2][t][m]
                        temp_arry1.append(form1)

                elif ModeA[j2][0] != [0, 1, 2, 3]:
                    for m in ModeA[j2][0]:
                        form1 = (t + ES.iat[j4]
+ D.iat[j4, t + ES.iloc[j4]][m]) *
var[j2][t][0]
                        temp_arry1.append(form1)

            for t in range(LS.iat[j3] - ES.iat[j3] + 1):
                if ModeA[j1][0] == [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form2 = var[j1][t][m]
* ((t + ES.iat[j3]) + D.iat[j3, t +
 ES.iloc[j3]][m] + lag - 1)
                        temp_arry2.append(form2)
                elif ModeA[j1][0] != [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form2 = var[j1][t][0]
* ((t + ES.iat[j3]) + D.iat[j3, t +
ES.iloc[j3]][m] + lag - 1)
                        temp_arry2.append(form2)

            prob += (pulp.lpSum(temp_arry1)
 >= pulp.lpSum(temp_arry2))
```

```
    # Create constraint 4: Start to start

    for i in range(C9.shape[0]):
        # print(P["element id"].iat[i])
        lag = int(C9["lag"].iat[i])
        if int(C9["pred"].iat[i] - 1) in BlistC
and int(C9["succ"].iat[i] - 1) in BlistC:
            j1 = BlistC.index(int(C9["pred"].iat[i] - 1))
            j2 = BlistC.index(int(C9["succ"].iat[i] - 1))
            j3 = int(C9["pred"].iat[i]) - 1
            j4 = int(C9["succ"].iat[i]) - 1
            temp_arry1 = []
            temp_arry2 = []
            for t in range(LS.iat[j4] - ES.iat[j4] + 1):
                if ModeA[j2][0] == [0, 1, 2, 3]:
                    for m in ModeA[j2][0]:
                        form1 = (t + ES.iat[j4]) * var[j2][t][m]
                        temp_arry1.append(form1)

                elif ModeA[j2][0] != [0, 1, 2, 3]:
                    for m in ModeA[j2][0]:
                        form1 = (t + ES.iat[j4]) * var[j2][t][0]
                        temp_arry1.append(form1)

            for t in range(LS.iat[j3] - ES.iat[j3] + 1):
                if ModeA[j1][0] == [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form2 = var[j1][t][m] *
(t + ES.iat[j3] + lag)
                        temp_arry2.append(form2)
                elif ModeA[j1][0] != [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form2 = var[j1][t][0] *
(t + ES.iat[j3] + lag)
                        temp_arry2.append(form2)

            prob += pulp.lpSum(temp_arry1)
>= pulp.lpSum(temp_arry2)

    # Create constraint 5: Start to finish

    for i in range(C10.shape[0]):
        # print(P["element id"].iat[i])
        lag = int(C10["lag"].iat[i])
        if int(C10["pred"].iat[i] - 1) in BlistC
and int(C10["succ"].iat[i] - 1) in BlistC:
            j1 = BlistC.index(int(C10["pred"].iat[i] - 1))
            j2 = BlistC.index(int(C10["succ"].iat[i] - 1))
            j3 = int(C10["pred"].iat[i]) - 1
            j4 = int(C10["succ"].iat[i]) - 1
            temp_arry1 = []
            temp_arry2 = []
            for t in range(LS.iat[j4] - ES.iat[j4] + 1):
                if ModeA[j2][0] == [0, 1, 2, 3]:
                    for m in ModeA[j2][0]:
                        form1 = (t + ES.iat[j4]
+ D.iat[j4, t + ES.iloc[j4]][m] - 1)
* var[j2][t][m]
                        temp_arry1.append(form1)

                elif ModeA[j2][0] != [0, 1, 2, 3]:
                    for m in ModeA[j2][0]:
                        form1 = (t + ES.iat[j4]
+ D.iat[j4, t + ES.iloc[j4]][m] - 1)
* var[j2][t][0]
                        temp_arry1.append(form1)

            for t in range(LS.iat[j3] - ES.iat[j3] + 1):
                if ModeA[j1][0] == [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form2 = var[j1][t][m]
* (t + ES.iat[j3] + lag)
                        temp_arry2.append(form2)
                elif ModeA[j1][0] != [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form2 = var[j1][t][0]
* (t + ES.iat[j3] + lag)
                        temp_arry2.append(form2)

            prob += (pulp.lpSum(temp_arry1)
 >= pulp.lpSum(temp_arry2))

    # Create constraint 9: finish on or after

    for i in range(C4.shape[0]):
        # print(P["element id"].iat[i])
        j3 = int(C4.iloc[i, 1])
        if int(C4.iloc[i, 0] - 1) in BlistC:
            j1 = BlistC.index(int(C4.iloc[i, 0] - 1))
            j2 = int(C4.iloc[i, 0] - 1)
            temp_arr = []
            for t in range(LS.iat[j2] - ES.iat[j2] + 1):
                if ModeA[j1][0] == [0, 1, 2, 3]:

                    for m in ModeA[j1][0]:
                        form1 = var[j1][t][m]
* (t + ES.iat[j2] + D.iat[j2, t +
ES.iloc[j2]][m] - 1)
                        temp_arr.append(form1)

                elif ModeA[j1][0] != [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form1 = var[j1][t][0]
* (t + ES.iat[j2] + D.iat[j2, t +
 ES.iloc[j2]][m] - 1)
                        temp_arr.append(form1)

            prob += (pulp.lpSum(temp_arr) >= j3)

    # Create constraint 10: Start on or before

    for i in range(C5.shape[0]):
        # print(P["element id"].iat[i])

        j3 = int(C5.iloc[i, 1])
        if int(C5.iloc[i, 0] - 1) in BlistC:
            j1 = BlistC.index(int(C5.iloc[i, 0] - 1))
            j2 = int(C5.iloc[i, 0] - 1)
            temp_arr = []
            for t in range(LS.iat[j2] - ES.iat[j2] + 1):
                if ModeA[j1][0] == [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form1 = var[j1][t][m]
 * (t + ES.iat[j2])
                        temp_arr.append(form1)

                elif ModeA[j1][0] != [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form1 = var[j1][t][0]
* (t + ES.iat[j2])
                        temp_arr.append(form1)

            prob += (pulp.lpSum(temp_arr) <= j3)

    # Create constraint 11: finish on or before

    for i in range(C6.shape[0]):
        j3 = int(C6.iloc[i, 1])
        if int(C6.iloc[i, 0] - 1) in BlistC:
            j1 = BlistC.index(int(C6.iloc[i, 0] - 1))
            j2 = int(C6.iloc[i, 0] - 1)
            temp_arr = []
            for t in range(LS.iat[j2] - ES.iat[j2] + 1):
                if ModeA[j1][0] == [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form1 = var[j1][t][m] *
(t + ES.iat[j2] + D.iat[j2,
t + ES.iloc[j2]][m] - 1)
                        temp_arr.append(form1)

                elif ModeA[j1][0] != [0, 1, 2, 3]:
                    for m in ModeA[j1][0]:
                        form1 = var[j1][t][0] *
(t + ES.iat[j2] + D.iat[j2, t
+ ES.iloc[j2]][m] - 1)
                        temp_arr.append(form1)

            prob += (pulp.lpSum(temp_arr) <= j3)
    #Create constraint 12: resource availability

    for k in range(int(R)):
        for s in range(int(H + 1)):
            temp_arr = []
            for j in range(len(BlistC)):
                if ModeA[j][0] == [0, 1, 2, 3]:
                    for m in ModeA[j][0]:
                        for t in range(int(max(s -
DB.iloc[BlistC[j], s][m] - 1  +
AC.iloc[int(SGS["calendar"].iat[BlistC[j]]),
 s - DB.iloc[BlistC[j], s][m]]+
 AC.iloc[int(SGS["calendar"].iat[BlistC[j]]),
s - DB.iloc[BlistC[j], s][m] -
1], ES[BlistC[j]]) - ES[BlistC[j]]),
int(min(LS[BlistC[j]], s) - ES[BlistC[j]] + 1)):
                            form1 = RD.iloc[BlistC[j], k]
* var[j][t][m] * AC.iloc[
 int(SGS["calendar"].iat[BlistC[j]]), s]
                            temp_arr.append(form1)

                elif ModeA[j][0] != [0, 1, 2, 3]:
                    for m in ModeA[j][0]:
                        for t in range(int(max(s -
DB.iloc[BlistC[j], s][m] - 1 +
AC.iloc[int(SGS["calendar"].iat[BlistC[j]]),
 s - DB.iloc[BlistC[j], s][m]] +
 AC.iloc[int(SGS["calendar"].iat[BlistC[j]]),
s - DB.iloc[BlistC[j], s][ m] - 1]
, ES[BlistC[j]]) - ES[BlistC[j]]), int(
min(LS[BlistC[j]], s) - ES[BlistC[j]] + 1)):
                            form1 = RD.iloc[BlistC[j], k]
```

```
* var[j][t][0] * AC.iloc[
int(SGS["calendar"].iat[BlistC[j]]), s]
                            temp_arr.append(form1)

            prob += (pulp.lpSum(temp_arr)
<= AV.iat[k, s])

    # # prob.solve()
    prob.solve(GUROBI_CMD(mip=3, options=[("TimeLimit", 1500),
("Heuristics", 0.2), ("Symmetry", 2)], warmStart=False))
```

Appendix F

Python Code of Set Project Start Dates to Optimized Ones

```
# save ES, LS, ES r decision variables
for j in range(len(BlistC)):
    for t in range(len(var[j])):
        for m in range(len(var[j][t])):
            if var[j][t][m].varValue >= 0.9:
                # if not SGS["ES"].iat[BlistC[j]]
== SGS["LS"].iat[BlistC[j]]:

                SGS["ES"].iat[BlistC[j]] = t + ES.iloc[BlistC[j]]
                SGS["ES r"].iat[BlistC[j]] = t + ES.iloc[BlistC[j]]
                SGS["LS"].iat[BlistC[j]] = t + ES.iloc[BlistC[j]]

                ES = pd.Series(SGS["ES r"].values)
                ES = ES.astype('int')
                LS = pd.Series(SGS["LS"].values)
                LS = LS.astype('int')
                # PR = pd.Series(SGS["Priority"].values)
                # PR = PR.astype('int')

# recreate Mode list
del ModeA[:]

# update decision variables modes
 for j in range(len(BlistC)):
     K = 0
     for t in range(len(var[j])):
         T = 0         for m in range(len(var[j][t])):

             J = 0
             if var[j][t][m].varValue >= 0.9:
                 M = []
                 M.append(m)
                 ModeA.append([M])

                J = 1
            T = T + J
         K = K + T

for j in range(len(BlistC)):
    K = 0
    for t in range(len(var[j])):
        T = 0
        for m in range(len(var[j][t])):
            J = 0
            if var[j][t][m].varValue >= 0.9:
                M = []
                M.append(m)
                ModeA.append([M])
                J = 1

            T = T + J
        K = K + T
    if K != 1:
        M = []
        for i in range(q):
            M.append(i)
        ModeA.append([M])

# Store t value
jValue = []
tValue = []

for j in range(len(var)):
    for t in range(len(var[j])):
        for m in range(len(var[j][t])):
            if var[j][t][m].varValue >= 0.9:
                numbers = re.findall(r’\d+’, var[j][t][m].name)
                Act = int(numbers[0]) - 1
                numbers = int(numbers[1])
                jValue.append(Act)
                tValue.append(numbers)

# Use resource
for j in range(len(BlistC)):
    for m in ModeA[j][0]:
        for k in range(int(R)):
            for s in range(tValue[j], tValue[j] + D.iat[jValue[j],
 ES.iloc[jValue[j]]][m] - 1):
                AV.iat[k, s] = AV.iat[k, s] - (
                        RD.iloc[jValue[j], k] * AC.iloc[
int(SGS["calendar"].iat[jValue[j]]), s])
```