TWO PHASED OPTIMIZATION APPROACH FOR SCHEDULING PROJECTS IN A MACHINE SHOP

by

Milad Akrami

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

at

Dalhousie University
Halifax, Nova Scotia
August 2015

This thesis is dedicated to my parents and my lovely fiance.For their endless love, support and encouragement.

Table of Contents

List of Tables

List of Figures

Abstract

This thesis, based on an industry problem, looks at the job-shop scheduling problem from a project planning and scheduling perspective. In the job-shop in question, a client customer order or project involves multiple jobs, each job consisting of batches. The job-shop has two types of resources, namely operators and machines and involves generalized precedence relations arising from part routings. In addition, the job-shop environment has other considerations such as project priority, alternative resources, and partial resource usage.

An integer linear programming with binary variables is developed for the job-shop with the above features in mind considering two objectives, with the minimization of total weighted tardiness as the primary objective and minimization of weighted throughput time for all projects as the secondary objective. This model can be used at two different hierarchical production planning levels, namely for high level planning and detailed scheduling simply by changing the time-scale of the model and passing the constraints of a high level planning solution to the lower level detailed scheduling problem. The higher level planning of the model is aimed at upper management in the job-shop who can use it for aggregate purposes such as resource allocation, customer order promising, due-date planning, and material procurement. On the other hand, the detailed scheduling is targeted towards production managers and is used to develop detailed short-term operation schedules.

The model is coded in Pulp, a linear/integer programming modeler written in Python, which can call a variety of solvers. Several numerical experiments are run on example problems of various sizes. The model was also solved using real data from the job-shop to show how the model can be used flexibly for both planning and scheduling, while meeting the functional requirements of the job-shop. The computational time to solve the problem was reasonable and the solution time characteristics are presented.

List of Abbreviations Used

EDD	Earliest Due Date
FJSSP	Flexible Job Shop Scheduling Problem
GA	Genetic Algorithm
ILP	Integer Linear Programming
JSSP	Job Shop Scheduling Problem
MTO	Make-To-Order
OSSP	Open Shop Scheduling Problem
PM	Project Management
RCPSP	Resource Constraint Project Scheduling Problem
SPT	Shortest Processing Time

Acknowledgements

I would like to thank my supervisor Dr. Alireza Ghasemi for providing me with the perfect balance of guidance, support, continuous encouragement, and friendship throughout this research. This thesis would not have been possible without his invaluable advice in all stages of the work.

Thank you to Dr. Mahdi Tajbakhsh for co-supervising this project.

A special thanks to Dr. Uday Venkatadri for his great collaboration during this research.

Thank you to Dr. Jing Chen for offering her time and knowledge in evaluating my thesis.

I would like to give my heartfelt appreciation to my parents, who brought me up with their love and encouraged me to pursue advanced degrees. Also, I would like to give my heartfelt appreciation to my fiance, who has accompanied me with her love, unlimited patience, understanding, helping and encouragement. Without her support, I would have never been able to accomplish this work.

Chapter 1

Introduction

Project Management (PM) is the process and activity of planning, organizing, motivating, and controlling resources, procedures and protocols to achieve specific goals. One of the vital issues of PM is the determination of the project schedule when the resources required are limited which is referred to as Resource Constraint Project Scheduling Problem (RCPSP). An example of resource-constrained project scheduling applied in manufacturing systems is job-shop scheduling. The project completion time is governed by the resource constraints and precedence relationships between the jobs within the project.

In a typical job-shop environment, customers arrive dynamically with a request for an order. The shop management and the customer negotiate the order specifications and its delivery date. If the customer decides to place the order, it is entered to the shop as a project with a specific due-date. In job-shop, it is customary to think of different orders as projects. Prior to planning and detailed scheduling of a project, the specific machining requirements are determined then the project is scheduled depending on the due-date, availability of resources, and materials [25].

The Job Shop Scheduling Problem (JSSP) is one of the most complex problems in scheduling and has been studied since the 1950s. A project consists of a number of jobs to be performed according to a set of precedence relations. All jobs have a processing time and usually require resources for their execution [6]. The aim of job-shop scheduling is therefore to assign jobs to resources in order to optimize an objective function such as minimization of total weighted tardiness or minimization of throughput time. Late projects are subjected to a predefined penalty per unit of lateness depending on the importance of the projects. Weighted tardiness is the sum of the tardiness of each project multiplied by the associated penalty. Throughput time is the total time of all

projects in the shop [21].

The JSSP is a very relevant problem in today's competitive environment where manufacturing companies are very concerned about their ability to quote accurate due-dates, organize efficient production, and ensure timely delivery. Projects must be delivered according to promised due-dates to satisfy customer expectations. While the loss of customer goodwill due to late delivery can be thought of as a penalty, penalties are sometimes explicitly written into project contracts.

The JSSP has gained much attention from researchers over the last 6 decades and the literature on the subject is vast. There are many different approaches to solve the problem. Since the JSSP is NP-hard (non-deterministic polynomial-time hard), many researchers have focused on feasibility and near-optimality. Since the JSSP needs to be solved continuously and repeatedly, the speed of the solution algorithm is important. The solution time (particularly in optimal approaches) depends on the network complexity of the JSSP. According to [7], in a classic JSSP, network complexity is a function of the number of jobs, the number of precedence relationships, the length of the time horizon, and the proximity of due-dates (shorter due-dates lead to more difficulty problems). Additional constraints and requirements such as batching could increase the network complexity.

1.1 Thesis Objective

The purpose of this thesis is to develop a planning and scheduling solution for a tool and die manufacturing company operating as a job-shop environment to improve its productivity and increase customer satisfaction. As the company grows and the number of employees continues to rise, it becomes more cumbersome for an individual to manage orders manually and efficiently. In addition, the environment is dynamic and Make-To-Order (MTO), which makes it even more difficult to predict and manage the workload in the shop. As a consequence, production in the shop is in constant flux in order to accommodate due-dates and priorities, which is one of the most difficult aspects of managing projects through their entire life-cycle (from the projects entering time to the

delivery time). The job-shop in question has the basic elements of the JSSP, namely, multiple resource constraints and generalized precedence relations as well as several requirements that go beyond the classic JSSP.

The objective of this thesis is to build a production scheduling system that considers resource availability, order due-dates, and order priorities. The objectives considered in this thesis are minimizing the total weighted tardiness and weighted throughput time. While heuristics can often provide a satisfactory solution in reasonable computational time, it was felt that with advances in problem formulation, optimization software, and computer hardware, that an exact optimization approach would provide a good benchmark for initial implementation. In the thesis, the focus is on exact optimization methods.

In conclusion, this thesis presents a planning and scheduling framework for a job-shop. The optimization model, its use in both planning and scheduling, and application in real-life are presented. Furthermore, the performance of the optimization model is contrasted against typical job-shop heuristics such as Shortest Processing Time (SPT), Earliest Due Date (EDD), SPT-EDD, and EDD-SPT.

1.2 Thesis Organization

The remainder of this thesis is organized as follows: A literature review relevant to project and job-shop scheduling problem is presented in chapter 2. In chapter 3, the exact problem is discussed in greater detail. In chapter 4, an Integer Linear Programming (ILP) formulation to solve the problem is presented. In chapter 5, the results of implementing the ILP model are presented and compared against classic job-shop heuristic approaches, concludes the thesis and identifies areas for future research.

Chapter 2

Literature Review

In this chapter, a review of the RCPSP and the JSSP is presented . This review focuses on problem extensions in both cases. First, research papers are classified based on their problem context and then discussed based on methodology.

2.1 Project Scheduling With Limited Resources

In the RCPSP, a project consist of a set of jobs. The jobs are interrelated by two types of constraints: (1) precedence constraints; which force job j to be scheduled after all of its predecessors are completed, and (2) resource requirements [20]. The basic RCPSP assumes non-preemptive jobs, meaning that no interruption is allowed once a job has started. Each job requires the use of one or more resources during its execution. Resources can only perform one job at a time. The resource requirements of a job are constant during processing and are renewable fully or partially in every period. In real-life situations, delays in the execution time of jobs occur when resources (e.g. machine or operator) required by these jobs are not available during the time interval when they are scheduled to take place. The release date is the earliest time at which a project may be started (it may be constrained by the availability of materials required for the project). Likewise, the due-date is the latest time a project must be finished [15].

The RCPSP is considered in [5], [2], [17] and [20] with different solution methodologies. In [5], a mathematical formulation with binary variables is proposed. The formulation exploits three variables, one associated with the start time of a job, one associated with the finish time of a job, and the last one associated with the portion of a job in progress at a given time. This approach resulted in lower computation time compared with other approaches in the literature.

In [17], two new mixed integer linear programming methods are proposed for solving the RCPSP. The two methods are start/end event-based formulation and on/off event-based formulation. The main advantage of these formulations is that fewer variables are needed to formulate the problem. It is claimed that this approach reduces the computation time because the variables are event-based as opposed to time-based and they are not a function of the time horizon. The advantage of these methods is seen in cases with long scheduling horizons. In contrast, according to the state of art presented in [19], in order to account for resource limitations other than processing units, the unit-specific-time-event based formulation requires a new set of constraints and variables which monitor the level of resources at every time event. Because the same time event can take place at different times for different units, these constraints are significantly more complex and numerous than in the case of global time points. A larger number of event points, as well as additional continuous variables for timing of resources, are needed.

In [11], a branch and bound method is proposed for solving RCPSP with generalized precedence relations. The method aims at an optimum solution by minimizing the project makespan. Makespan is the time by which all projects are completed [21], in other words, it is the completion time of the last project. The RCPSP is extended to include arbitrary minimal and maximal time lags between the starting and completion times of jobs. In the minimal lag, a job can start (finish) if the predecessor has already begun (finished) after a particular period. In the maximal lag, a job should be started (finished) at the latest a particular period beyond the start (finish) of another job. The proposed extension allows to model very general class of project scheduling problems including:

- Precedence constraints.
- Job arrival times and due-dates.
- Multiple resource constraints with time-varying resource requirements and availabilities.
- Job and resource time windows and several types of mandatory job overlaps.

Genetic Algorithm (GA) is a problem-solving technique that uses the concepts of evolution and heredity to produce good solutions to complex problems that typically have enormous search spaces and are therefore difficult to solve. In [2], a differential evolution algorithm is proposed based on the GA meta-heuristic. This approach provides similar results when compared with existing approaches. In [20], a multi-array project-oriented GA formulation is used to represent chromosomes that provide similar solution quality as in the literature with lower computation time.

The single and multi-mode RCPSPs are considered in [6] who propose a simulated annealing algorithm to solve the problem . The proposed algorithm handles different objective functions including the minimization of makespan, total project cost, and the maximization of net-present value. In the single mode problem, there is only one duration and resource requirement for every job. In the multi-mode problem, there are several modes, i.e., processing time and resource requirement, for every job. A start time and an execution mode must be chosen for each job.

In [12], a local search methodology and a Tabu search procedure are proposed for solving the RCPSP. In this methodology, the proposed approach in [5] (multiple execution modes of jobs) and [11] (minimal and maximal time lags between the starting and completion time of a job) are combined. Jobs have multiple execution modes. It is concluded that a more efficient way of using resources can be achieved because multi-modal problems allow trade-offs between time and cost, depending on resource usage.

The RCPSP with several multi-mode projects allowing for renewable and non-renewable resources is considered in [4]. The modeling approaches in this paper for multi-project scheduling do not allow resource sharing between the projects. This multi-project problem environment is defined as the Resource Dedication Problem (RDP). RDP seeks the optimal dedication of resource capacities to different projects within the overall limits of the resources and with the objective of minimizing a predetermined objective function. This paper proposes two methodologies: (1) GA employing a new improvement move called combinatorial auction for RDP, which is based on preferences of projects for resources, and (2) a linear and Lagrangian relaxation.

In [3] a RCPSP with preemptive jobs is considered where a maximum of one interruption

per job is allowed. The proposed heuristics in the literature are modified to deal with preemptive jobs. The paper concludes that preemption significantly helps decrease project length.

2.2 Job Shop Scheduling With Limited Resources

The classic JSSP is considered in [1], [8], [26] and [18] with different solution methodologies. In [1], a shifting bottleneck procedure is proposed. This heuristic method sequences one machine at a time. At first, it solves the one-machine scheduling problem to optimality. Then, it uses the outcome for ranking and sequencing the machines with the highest priority. Every time a new machine is sequenced, the heuristic checks for the machines that are already sequenced. This procedure repeats until all the machines are optimally sequenced. This paper solves the famous 10*10 problem with 10 machines and 10 jobs [**?**].

A branch and bound method is used for solving the JSSP in [8]. In this approach, a generalization of a branching scheme and a method to fix disjunction before each branching are combined. The objective function for this problem is the minimization of the makespan. This method is used to solve the 10*10 benchmark problem. The proposed approach solves the problem faster than the proposed approach in [1]. In [26] the JSSP is solved using GA. Three algorithms are presented and the proposed algorithms find feasible solutions within reasonable computational time. In [18], a Tabu thresholding heuristic that uses a new block-based neighborhood function is proposed to solve the JSSP with limited machine availability.

A Tabu thresholding consists of the successive iteration between intensification phase called improving phase, and diversification phase called mixed phase. The improving phase is the steepest descent that iteratively searches for a local optimum by visiting neighbor solutions that are generated by improving moves. Once the neighborhood does not contain any improving move, the intensification phase is stopped, and the mixed phase is started from the local optimum. The mixed phase is a diversification that accepts both improving and non-improving moves to escape from local optima and thus to guide the search toward new regions. The objective function for this approach

is the minimization of the makespan. A concept of blocks is used to consider the unavailability periods of machines. Hence, several efficient neighborhood functions are defined using the concepts of blocks. This approach does not promise a better solution as opposed to the ones proposed in the literature.

Project priority in JSSP is considered in [25], [13], [27], [28], [23] and [21]. Project priority determines which competing projects can access limited resources (operator or machine) faster. Project priority is assigned according to customer importance or project due-date. A general two-step approach for solving the problem is proposed in [25]. The heuristic approach essentially decomposes the problem into two easier problems. First, the proposed approach suggests the release and sequencing of jobs to minimize work in process (WIP). Then, due-dates are set to reduce the due-date lead time (due-date minus arrival time). A simulation experiment is performed to demonstrate the strength of the approach compared to other research. In [27], the Particle Swarm Optimization (PSO) technique is proposed. PSO is a type of intelligent optimization algorithm developed from swarm intelligent optimization. The PSO algorithm searches for the best solution over the complex space through co-operation and competition. This efficient approach is said to provide good solutions.

In [13], a GA approach is proposed. This approach is used to address the Open Shop Scheduling Problem (OSSP) and the job-shop rescheduling problem. OSSP is an extention of JSSP where there is no priority among the jobs within a project. Job-shop rescheduling is a process of revising the estimated processing time or the start time of some jobs. This approach improves the result obtained from other GA methods in literature.

According to [28], the JSSP and due-date assignments are two interrelated problems (e.g. a tighter due date setting will increase the chances of tardiness despite its appeal for the incoming customer). Hence, this paper integrates the problems as one optimization problem to achieve a better solution. A double-layered heuristic approach is proposed to solve the integrated problem. In the first layer, a GA is used to assign rough due-dates to unassigned jobs. The objective of this approach is to minimize the total weighted tardiness of all jobs. Then, the parameter perturbation method is employed to optimize the due-dates based on their potential interplay in the second layer.

The relation between a job's due-date is important because the due-date for one job may have an impact on the optimal due-date of other jobs. The proposed approach is said to be efficient and has the potential for future extensions.

Also, duration uncertainty in JSSP is considered in [28] and [23]. In this approach, a factor is assigned to deal with unexpected events in processing times. Alternative resources are also considered in [23] and [21]. In the presence of alternative resources, one or more resources are capable of performing a job.

Flexible Job Shop Scheduling Problem (FJSSP) is an extension of the JSSP, where jobs are allowed to be processed on the multiple available resources. Different approaches are proposed in [9], [29] and [22] for solving the FJSSP. In [9] an iterative GA method is proposed. This algorithm indicates the potential improvement in FJSSP, especially with multiple identical machines under a resource-constrained environment. The proposed model helps in quantifying the flexibility between machines under limited resources. Two approaches for solving the FJSSP are proposed in [29]. The first approach is a hierarchical method based on two steps: (1) assigning jobs to machines, and (2) job sequencing, which determines the order of jobs on the different machines. Minimizing total tardiness is the objective for both steps. The second approach is the integration of the two steps of the first method that is based on GA.

In [22], a greedy randomized adaptive search procedure (GRASP) is proposed to solve the FJSSP. The proposed meta heuristic algorithm consists of two stages: (1) construction and (2) local search. The objective function for this approach is to minimize the makespan and maximize total workload. Three problems instances (8*5, 12*5 and 8*8) are used to demonstrate the strength of the approach. The first two cases provides better result compared with another paper in the literature. However, last problem could not be compared with other approaches because it was not solved before. The balance between resource limitation and machine flexibility is also discussed in this paper.

Splitting in the JSSP is considered in [23] and [21]. In splitting, jobs can be broken down to a number of batches, and different resources can be allocated to do each batch. Batches may or may not start at the same time. Splitting is very flexible from the scheduling point of view, but the set-up cost dictates the extent of splitting [21]. While

GA is used in [23] to conclude that the schedule can be improved using splitting, this paper does not use setup costs.

In [21], a zero-one integer linear programming approach is proposed to solve the JSSP with limited resources. The extensions discussed in this paper are alternative resources, splitting, concurrency and non-concurrency of jobs. In alternative resources, one or more operators or machines are capable of performing a job. Concurrency and non-concurrency of jobs state that two jobs must or must not be performed simultaneously. In non-concurrency, jobs are allowed to be performed in any order. Three objective functions are considered in the paper including, the minimization of total throughput time , the minimization of makespan and the minimization of total tardiness for all projects. Individual project throughput time is defined as the elapsed time between project arrival time and project completion time.

2.3 Discussion and Conclusion

Project scheduling is mostly concerned about projects in the context of construction and shipbuilding, while job-shop scheduling is applied to scheduling problems in job-shops. A summary of the literature review is made through three tables (2.1, 2.2, 2.3), which map and classify project and job-shop scheduling problems with limited resources.

In Table 2.1, the papers in the literature are classified according to solution methodology and problem characteristics. As can be seen in the table, most papers consider precedence relations, while some consider project priority. A few papers also consider other extensions such as alternative resources, duration uncertainty, preemption and splitting. Also, most problem extensions explained above are considered in [23] and [21], while a majority of the reviewed papers use heuristics or meta heuristics.

The two other tables organize the same information differently: Table 2.2 shows the terms used in the objective function of the models in the paper depending on the characteristics of the problem considered. Finally, Table 2.3 maps the methodology used in the papers with the different terms used in the objective function.

In partial resource usage, some jobs may not require full resource capacity. Hence,

Table 2.1: Methodologies vs. Problem Characteristics

		Problem Characteristics							
		Job-Shop Scheduling	FJSSP	Precedence relations	Project Priority	Preemption	Alternative Resources	Duration uncertainty	Splitting
Methodologies	Linear Programming	21		21,5	21	21	21		21
	Simulation	25		25	25				
	Branch and Bounds	8		8,11,16					
	Simulated Annealing			6					
	Shifting bottleneck procedure	1		1					
	Tabu search	18		12,18	12				
	Hierarchical method and Genetic Algorithm								
	Heuristics			3		3			
	Particle swarm optimizer	27		27	27				
	MILP			17					
	Genetic Algorithm	13,26,9,23 ,28	9	13,26,9,23,20,4	13,23,20,28	4	9,23	23	23
	A metaheuristic greedy randomised adaptive search procedure	22	22	22			22		
	Parameter perturbation	28			28			28	
	Lagrangian relaxation			4		4			
	Differential evolution algorithm			2	2				

Table 2.2: Objective Functions vs. Problem Characteristics

		Problem Characteristics							
		Job-Shop Scheduling	FJSSP	Precedence relations	Project Priority	Preemption	Alternative Resources	Duration uncertainty	Splitting
Objective Function	Min makespan	13,8,1,23,2 6,27,22	22	13,8,11,16,1,12,3,2 3,20,5,2,25,26,6,22	13,23,20,2, 25	12,3	22,23	23	23
	Min Total Weighed Tardiness	28		4	28	4		28	
	Minimize Makespan & Machine Idle Cost	9	9	9			9		
	Multi-objectives	21		21,17	21	21	21		21
	Min Tardiness	29	29	29			29		
	Max & Total Workload	22	22	22			22		
	Min WIP & Due-date Lead Time	25		25	25				

Table 2.3: Methodologies vs. Objective Functions

		Objective Function						
		Min makespan	Min Total Weighed Tardiness	Machine Idle Cost	Multi-objectives	Min Tardiness	Max & Total Workload	Min WIP & Due-date Lead Time
Methodologies	Linear Programming	5				21		
	Simulation							25
	Branch and Bounds	8,11,16						
	Simulated Annealing	6						
	Shifting bottleneck procedure	1						
	Tabu search	12,18						
	Hierarchical method and Genetic Algorithm					29		
	Heuristics	3						
	Particle swarm optimizer	27						
	MILP				17			
	Genetic Algorithm	13,23,20,26,9	28,4	9				
	A metaheuristic greedy randomised adaptive search procedure	22					22	
	Parameter perturbation		28					
	Lagrangian relaxation		4					
	Differential evolution algorithm	2						

resources may be assigned to more than one job during a particular period to the extent of their full capacity. In batching, jobs can be broken down to a number of batches so that when these batches are completed, successor machines can start processing without starvation. Based on the papers considered in this literature review, partial resource usage and batching have not been considered in the past. In fact, one of the assumptions in the JSSP is that once a resource is assigned to a job, no other jobs can be allocated to it.

In [21], a solution is proposed to the JSSP with job priority, alternative resources, and splitting. This paper uses a binary linear programming approach to solving the optimization problem to obtain optimum solutions. The model was written a long time ago where lack of technology (old computers) forbade solving problems with reasonable sizes. This thesis uses some of the basic ILP formulations proposed in that paper to successfully solve much larger problems.

Chapter 3

Thesis Context and Assumptions

The problem considered in this thesis may be thought of as a RCPSP with generalized precedence relations, alternate resources, partial resource usage, batching, and project priorities.

The case study used in this thesis is a MTO job-shop which manufactures machined components on order. This strategy results in longer wait times for the consumer to receive the product but allows for more product customization compared to directly purchasing from product catalogs. In companies with the MTO production strategy, effective resource allocation significantly improves the production efficiency. At the job-shop, a PM handles the scheduling of jobs by taking into consideration the process plans required for machining, the availability of both machining and operator resources, and the preferred customer due-dates.

The process of resource allocation is often very difficult and time-consuming due to the combinatorial complexity of the decision space. The complexity of this process increases as the number of resources and projects increase, as is the case at the MTO job-shop which has been growing in recent years. Therefore, it was determined that a tool for optimal resource allocation that takes into account production constraints would be beneficial.

3.1 Definitions

Customers arrive at the job shop and negotiate a due-date for a given quantity of products (e.g. machined components or machined parts). Once a contract is signed with the customer, a project is created. Each product has an ordered set of processes. Figure 3.1 shows a project created for a product with three processes, where the dotted lines show the precedence relationships between the processes.

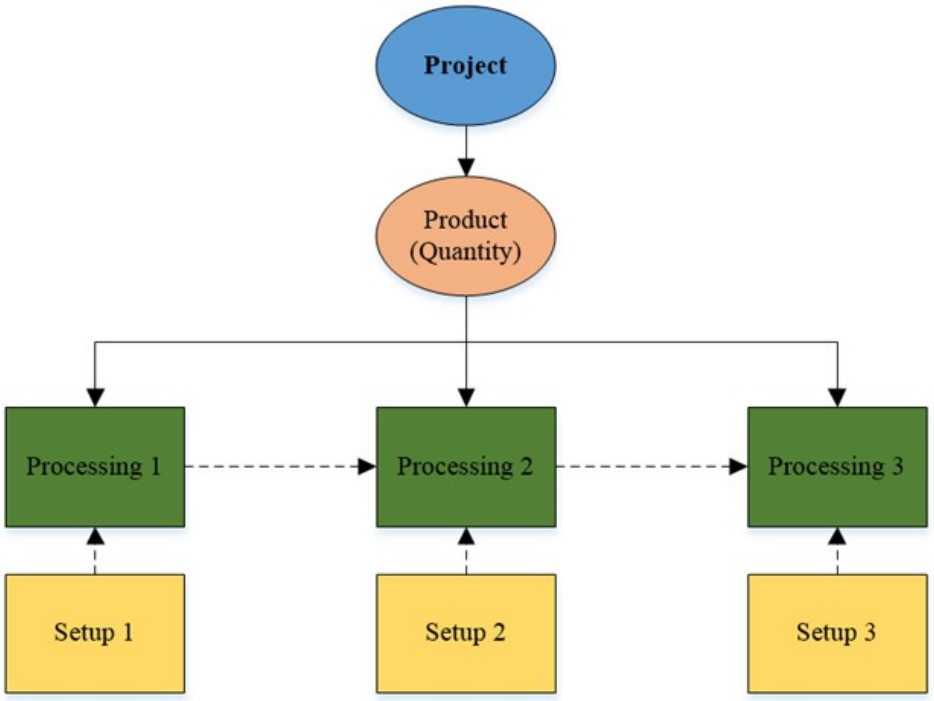


Figure 3.1: Hierarchical representation of a project

Also as shown in figure 3.1, there is a setup step for each of the processes. Based on these setup and processing requirements, a set of jobs with precedence relationships is created for each project, as shown in figure 3.2. In the figure, each process is effectively represented by two jobs: a setup job and a process job. Therefore, the example in the figure shows six jobs that cover the three processes. Without loss of generality, each process (a combination of a setup job and a processing job together) requires a machine (from an eligible list) and each job (whether setup or processing) requires an operator (from an eligible list). The required precedence relationships between setup jobs and processing jobs and between processing jobs is also shown in the figure.

Since the setup operation is carried out exactly once for an entire product batch, the duration of the setup jobs depends on the product and may depend on the machine assigned to the job. On the other hand, the duration or cycle time of the processing jobs depends on the required production quantity and may depend on the assigned machine. Therefore, for all active projects, the corresponding setup and processing jobs can be generated along with their durations and precedence requirements.

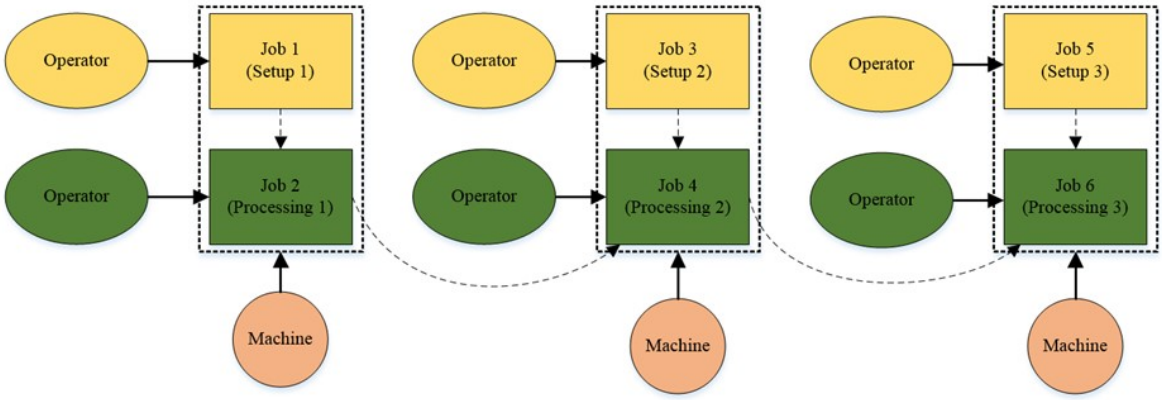


Figure 3.2: Setup and Processing Job Definitions

3.2 Thesis Assumptions

This section lists the common assumptions made in the JSSP, which are also made in this thesis:

- Projects have desired due-dates. No cost is incurred if the project is completed before the due-date. If it is completed after the due-date, a lateness cost is imposed. The lateness cost depends on the duration of lateness and the unit lateness cost may vary from project to project.
- Projects have absolute due-dates (latest possible finish time). Each project must be completed before its absolute due-date.
- Estimated setup and processing jobs durations are available.
- A processing job may have another processing job as predecessor. Predecessors

are the set of jobs that need to be completed before a job can start.

- Jobs (both setup and processing) are non-preemptive. This means that no interruption is allowed once a job is started.
- Machines/operators have limited capacities.
- A delay between the end of a setup job and the start of the corresponding processing job is permitted. Since the operator required for the setup may be different from the operator required for the processing job, it is sometimes advantageous to finish the setup job when the setup operator is available, even though processing job itself might start later.

The additional requirements specific to this thesis are:

- Machines and operators availability in each period may be different throughout the planning horizon.
- Processing jobs can be done in batches. Batching allows breaking down a processing job to enable successive jobs to start processing a batch as soon as it is available. This is done in a way to avoid starvation at a successor machine. Batching has no impact on the setup cost because a setup is incurred only once for a process sequence.
- Operators may be used partially for a job. In a real manufacturing environment, some jobs only need a portion of an operator's time. This concept is used to allow operators perform multiple jobs simultaneously. For example, an operator may be able to monitor two or three machines in process.
- One or more resources (machines or operators) may be capable of performing a job. A machine from the eligible list must be chosen for a setup and its corresponding processing job. On the other hand, an eligible operator must be chosen for the setup job and an eligible operator must be chosen for the corresponding processing job. Operators selected for the setup and processing jobs may or may not be the same, depending on their eligibility.
- An inflation factor may be used to incorporate the uncertainty in processing times.

- Projects are prioritized according to customer importance or upcoming due-dates. Penalty costs (in cost per unit time) for lateness are assigned to projects to reflect their priorities.

Chapter 4

Methodology

An ILP formulation with binary variables is developed to solve the aforementioned problem considering two objectives that minimize the total weighted tardiness and the weighted throughput time of all projects. The formulation contains multiple binary variables, two of which identify the completion time of the projects and jobs. Project completion time is dependent on the completion time of all its jobs, and contributes directly to the objective function. The job completion time is used to impose the precedence requirements as well as batching. Job processing periods are used to take into account the engagement of the resources while the job is being processed. What follows identifies the elements of the proposed ILP model in detail.

4.1 Mathematical Model

The definitions and variables used to develop the model are presented here followed by the model formulation.

4.1.1 Sets and Parameters

I	Set of projects
$\|I\|$	Total number of projects
J_i	Set of jobs in project i
$\|J_i\|$	Total number of jobs in project i
A_{ij}	Earliest possible time to start job j of project i
D_{ij}	Processing time required to perform job j of project i
ADD_i	Absolute due-date of project i
g_i	Desired due-date of project i
M	Set of machines
$\|M\|$	Total number of machines
O	Set of operators
$\|O\|$	Total number of operators
Opu_{ijo}	($<= 1$) Operator o requirement of job j of project i
l_{ij}	The earliest completion time of job j of project i in absence of resource constraints(Calculated in preprocessing stage)
u_{ij}	The latest completion time of job j of project i in absence of resource constraints (Calculated in preprocessing stage)
e_i	The earliest completion time of project i in absence of resource constraints (Calculated in preprocessing stage)
w_i	Priority of project i
B_{ij}	Total number of batches for job j of project i
R_{mt}	Capacity of machine m during period t
V_{ot}	Capacity of operator m during period t
ϵ	Weightage given to weighted throughput time objective, 0.01 by default

T Problem analysis horizon, starting from the minimum arrival of all projects to the their maximum absolute due-date.

T_i Set of feasible process periods for project i, starting from its earliest job start time to its absolute due-date, i.e., $T_i = \min_{\forall j \in J_i} (A_{ij}), \cdots, ADD_i \; \forall i \in I$

T_{ij} Feasible job completion times; set of periods starting from job j of project i's earliest possible completion time to the job's latest possible completion time. i.e., $T_{ij} = l_{ij}, \cdots, u_{ij}, \; \forall i \in I, \; \forall j \in J_i$

TEG_i Set of feasible project completion times; set of periods starting from project i's earliest possible completion time to the project's absolute due-date, i.e., $TEG_i = e_i, \cdots, ADD_i, \forall i \in I$

$$MR_{ijm} = \begin{cases} 1 & \text{If machine } m \text{ is } \textbf{capable} \text{ of performing machining requirement of job } j \text{ of project } i, \\ 0 & \text{Otherwise.} \end{cases}$$

$$OR_{ijo} = \begin{cases} 1 & \text{If operator } o \text{ is } \textbf{capable} \text{ of performing operator requirements of job } j \text{ of project } i, \\ 0 & \text{Otherwise.} \end{cases}$$

$$P_{ijk} = \begin{cases} 1 & \text{If } k \text{ is a precedence of } j \text{ in project } i, \\ 0 & \text{Otherwise.} \end{cases}$$

4.1.2 Variables

$$h_{it} = \begin{cases} 1 & \text{If project } i \text{ is completed at time } t, \\ 0 & \text{Otherwise.} \end{cases}$$

$$x_{ijt} = \begin{cases} 1 & \text{If job } j \text{ of project } i \text{ is completed at time } t, \\ 0 & \text{Otherwise.} \end{cases}$$

$$zz_{ijt} = \begin{cases} 1 & \text{If job } j \text{ of project } i \text{ is in process during period } t, \\ 0 & \text{Otherwise.} \end{cases}$$

$$zzs_{ijt} = \begin{cases} 1 & \text{If job } j \text{ (setup) of project } i \text{ is competed on a machine but its} \\ & \text{successor, job } j+1 \text{ (processing), has not started on the same} \\ & \text{machine yet,} \\ 0 & \text{Otherwise.} \end{cases}$$

$$S_{ijm} = \begin{cases} 1 & \text{If machine type } m \text{ is assigned to perform job } j \text{ of project } i, \\ 0 & \text{Otherwise.} \end{cases}$$

$$W_{ijo} = \begin{cases} 1 & \text{If operator type } o \text{ is assigned to perform job } j \text{ of project } i, \\ 0 & \text{Otherwise.} \end{cases}$$

$$QS_{ijmt} = \begin{cases} 1 & \text{If machine } m \text{ is utilized for job } j \text{ of project } i \text{ during period } t, \\ 0 & \text{Otherwise.} \end{cases}$$

$$QQS_{ijmt} = \begin{cases} 1 & \text{If the setup job } j \text{ related to processing job } j+1 \text{ has finished} \\ & \text{during period } t \text{ on machine } m, \text{ but the processing job } j+1 \\ & \text{has not yet started,} \\ 0 & \text{Otherwise.} \end{cases}$$

$$QW_{ijot} = \begin{cases} 1 & \text{For the assigned operator } o \text{ to job } j \text{ of project } i \text{ if the job is} \\ & \text{in process during period } t, \\ 0 & \text{Otherwise.} \end{cases}$$

4.1.3 Objective function

Minimize:

$$\sum_{i \in I} \sum_{t=g_i+1}^{ADD_i} [w_i * (t - g_i) * h_{it}] + \sum_{i \in I} \sum_{e_i}^{ADD_i} [\epsilon * w_i * h_{it} * t] \tag{4.1}$$

4.1.4 Constraints

Job Completion Time

$$\sum_{t=l_{ij}}^{u_{ij}} x_{ijt} = 1; \;\; \forall i \in I, \forall j \in J_i \tag{4.2}$$

Processing Periods

$$\sum_{s=t}^{(t+D_{ij})-1} x_{ijs} = zz_{ijt}; \qquad \forall i \in I, \; \forall j \in J_i, \; \forall t \in T_i \tag{4.3}$$

$$\sum_{s=l_{ij}}^{t} x_{ijs} - \sum_{p=l_{ij}}^{t+D_{ij}} x_{i,j+1,p} = zzs_{ijt}; \qquad \forall i \in I, \; \forall j \in J_i, \; \forall t \in T_i \tag{4.4}$$

Job Precedence

$$P_{ijk} * \left[\left(\sum_{t=l_{ijk}}^{u_{ik}} t * x_{ikt} \right) - D_{ik} + \frac{D_{ik}}{B_{ik}} \right] <= \sum_{t=l_{ijt}}^{u_{ij}} t * x_{ijt} - D_{ij}; \quad \forall i \in I, \forall j \in J_i, \forall k \in J_i \tag{4.5}$$

$$P_{ijk} * \left[\left(\sum_{t=l_{ik}}^{u_{ik}} t * x_{ikt} \right) + \frac{D_{ij}}{B_{ij}} \right] <= \sum_{l_{ij}}^{u_{ij}} t * x_{ijt}; \qquad \forall i \in I, \forall j \in J_i, \forall k \in J_i \tag{4.6}$$

Project Completion Time

$$\sum_{e_i}^{ADD_i} h_{it} = 1; \qquad \forall i \in I \tag{4.7}$$

$$\sum_{t=l_{ij}}^{u_{ij}} \sum_{j \in J_i} x_{ijt} >= (h_{is}) * |J_i|; \qquad \forall i \in I, \forall s \in TEG_i \tag{4.8}$$

Machine Assignment

$$\sum_{m \in M} S_{ijm} <= \sum_{m \in M} MR_{ijm}; \qquad \forall i \in I, \forall j \in J_i \tag{4.9}$$

$$|M| * \sum_{m \in M} S_{ijm} >= \sum_{m \in M} MR_{ijm}; \qquad \forall i \in I, \forall j \in J_i \tag{4.10}$$

$$\sum_{m \in M} S_{ijm} <= 1; \qquad \forall i \in I, \forall j \in J_i \tag{4.11}$$

$$S_{ijm} <= MR_{ijm}; \qquad \forall i \in I, \forall j \in J_i, \forall m \in M \tag{4.12}$$

$$0 <= zz_{ijt} + S_{ijm} - QS_{ijmt} * 2 <= 1; \qquad \forall i \in I, \forall j \in J_i, \forall m \in M, \forall t \in T \tag{4.13}$$

$$0 <= zzs_{ijt} + S_{ijm} - QQS_{ijmt} * 2 <= 1; \qquad \forall i \in I, \forall j \in J_i, \forall m \in M, \forall t \in T \tag{4.14}$$

$$S_{i,j+1,m} = S_{ijm}; i \in I, j \in J_i : mod(j, 2) = 1; \qquad \forall m \in M \tag{4.15}$$

Operator Assignment

$$\sum_{o \in O} W_{ijo} <= \sum_{o \in O} OR_{ijo}; \qquad \forall i \in I, \forall j \in J_i \tag{4.16}$$

$$|O| * \sum_{o \in O} W_{ijo} >= \sum_{o \in O} OR_{ijo}; \qquad \forall i \in I, \forall j \in J_i \tag{4.17}$$

$$\sum_{o \in O} W_{ijo} <= 1; \qquad \forall i \in I, \forall j \in J_i \tag{4.18}$$

$$W_{ijo} <= OR_{ijo}; \qquad \forall i \in I, \forall j \in J_i \tag{4.19}$$

$$0 <= zz_{ijt} + W_{ijo} - QW_{ijot} * 2 <= 1; \qquad \forall i \in I, \forall j \in J_i, \forall o \in O, \forall t \in T \tag{4.20}$$

Machine and Operator Capacity Restrictions

$$\sum_{i\in I}\sum_{j\in J_i} QS_{ijmt} + \sum_{i\in I}\sum_{j\in J_i} QQS_{ijmt} <= R_{mt}; \qquad \forall m \in M, \forall t \in T \tag{4.21}$$

$$\sum_{i\in I}\sum_{j\in J_i} QW_{ijot} * Opu_{ijo} <= V_{ot}; \qquad \forall o \in O, \forall t \in T \tag{4.22}$$

The objective function minimizes the total weighted tardiness and the weighted throughput time concurrently. The first part of the objective function in 4.1 considers a penalty for projects completed later than their desired due-date and is the primary objective. The penalty is weighted by project priority w_i. The second part penalizes the objective function for projects not being completed at the earliest possible time and is secondary objective; thus, the model minimizes the total weighted tardiness as the primary objective and minimizes the weighted throughput time for all projects as the secondary objective. The weightage for the second part of the objective function ($\epsilon = 0.01$ by default) may need to be adjusted based on the decision maker's preferences.

Constraint 4.2 forces job completion time to be between the earliest job completion time (l_{ij}) and the latest job completion time (u_{ij}). The summation on the LHS of this equation is set equal to 1, meaning each job can only be completed once during the specified periods.

Constraint 4.3 sets job processing period variable (zz_{ijt}) to 1 while job j is in process. The summation on the LHS of this equation is carried on all possible intervals equal to the length of the processing period of each job of all projects. This summation only produces value 1 if the job completion variable for project i of job j (x_{ijt}) is equal to 1 in one of the times during the interval. The job processing period variables are used to allocate resources considering their capacity during each period (see equations 4.13, 4.14, and 4.20).

Constraint 4.4 forces the job idling period variable (zzs_{ijt}) to be 1 while the completed job j (setup job) is waiting for job $j+1$ (processing job) to start. Job j and $j+1$ must be performed by the same machine. Once job j is completed, no other jobs can be allocated to the same machine unless job $j+1$ is completed. The LHS of the equation

is composed of the completion of setup job ($\sum_{s=l_{ij}}^{t} x_{ijs}$) and start of the processing job ($\sum_{p=l_{ij}}^{t+D_{ij}} x_{i(j+1)p}$). If $\sum_{s=l_{ij}}^{t} x_{ijs}$ is equal to 1 and $\sum_{p=l_{ij}}^{t+D_{ij}} x_{i,(j+1),p}$ is equal to 0 at time t, it means the setup job is completed and the processing job has not started yet, so zzs_{ijt} will be equal to 1. When the processing job starts being processed, the LHS of the equation, and consequently zzs_{ijt} will be equal to 0. The job idling period variable is used to occupy (idle) the machine allocated to the setup job when its value is equal to 1.

Constraints 4.5 and 4.6 jointly force precedence relations between jobs of a project and also incorporates the batching concept. If job j is a predecessor of job k, constraint 4.5 forces start time of job k to be after the completion time of the first batch of job j. The completion time of the predecessors first batch $\left[(\sum_{t=l_{ij_2}}^{u_{ik}} t * x_{ikt}) - D_{ik} + D_{ik}/B_{ik}\right]$ has to be before the start time of the successor. This constraints is only forced when P_{ijk} is equal to 1. Constraint 4.6 prevents interruptions (starvation) in successor job. It forces completion time of the successor ($\sum_{l_{ij}}^{u_{ij}} t * x_{ijt}$) to be after the predecessor completion time plus the time required to finish the last batch of the successor $\left[\left(\sum_{t=l_{ik}}^{u_{ik}} t * x_{ikt}\right) + D_{ij}/B_{ij}\right]$. Figure 4.1 shows the execution of jobs according to the concept of batching with respect to the above constraints.

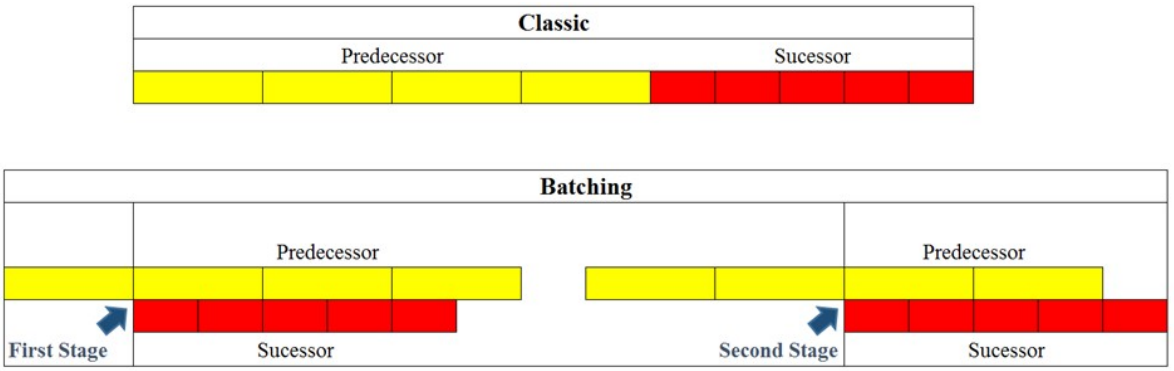


Figure 4.1: Classic execution of jobs comparing with batching concept

Constraint 4.7 forces the completion time of project i to be between earliest project completion time (e_i) and the latest project completion time (ADD_i). The summation on the LHS of this equation during the specified periods is equal to 1, meaning that projects can only be completed once. A project completion time is dependent on the completion time of all its jobs. Constraint 4.8 forces the completion time of project i to

be after the completion time of all its jobs. The LHS of the equation is the summation of the completion time of all jobs of project i, between the earliest and the latest job completion times (l_{ij} and u_{ij}). The project completion time (h_{is}) is multiplied by the total number of jobs of project $i, (|J_i|)$. This forces h_{is} to be equal to 1 when all the jobs of the project are completed.

Constraint 4.9 restricts the summation of the machines allocated to the job j, ($\sum_{m \in M} S_{ijm}$) to be less or equal to its total machining requirement, which is calculated in the RHS. If job j has no machining requirement, the RHS of the equation will be 0 which prevents allocation of any machine to the job. At least one machine must be allocated to the jobs with machining requirements. If job j requires a machine to perform it, the RHS of equation 4.10 will be equal or greater than 1. In the LHS of the equation, machine assignment variable (S_{ijm}) is multiplied by the total number of the machines ($|M|$) to force assigning at least one capable machine to the job. Constraint 4.11 prevents allocating more than one machine to a job regardless of its machining requirements. Constraint 4.12 forces the selection of the allocated machine to be a member of the set of capable machines to perform the jobs. Constraint 4.13 seizes the allocated machines during the processing period of jobs. If both job processing period (zz_{ijt}) and machine assignment variable (S_{ijm}) are equal to 1 during period t, machine assignment period (QS_{ijmt}) will become equal to 1, meaning that the machine is being utilized by job j during period t. Constraint 4.14 forces the allocated machine to setup job j to be idled while waiting for the processing job $j + 1$ to start being processed. During the idling period when zzs_{ijt} is equal to 1, one unit of the machine m capacity is occupied. If both job idling period (zzs_{ijt}) and machine assignment variable (S_{ijm}) are equal to 1 during period t, machine idling period (QQS_{ijmt}) will be equal to 1, meaning that the machine is being idled during period t. Constraint 4.15 forces the setup and processing jobs to be performed by the same machine.

Constraint 4.16 restricts the summation of the operators allocated to the job j ($\sum_{o \in O} W_{ijo}$) to be less or equal to its total operator requirements. If job j has no operator requirement, the RHS of the equation will be 0 which prevents allocation of any operator to the job. At least one operator must be allocated to jobs with operator requirement regardless of the number of operators capable of performing the job. If job j requires

an operator to perform it, the RHS of equation 4.17 will be equal or greater than 1. In the LHS of the equation, operator assignment variable (W_{ijo}) is multiplied by the total number of the operators ($|O|$) to force assigning at least one capable operator to the job. Constraint 4.18 prevents allocating more than one operator to a job regardless of its operator requirement. Constraint 4.19 forces the selection of the allocated operator to be a member of the set of capable operators to perform the jobs. Constraint 4.20 seizes the allocated operator during the processing period of jobs. If both job processing period (zz_{ijt}) and operator assignment variable (W_{ijo}) are equal to 1 during period t, operator assignment period (QW_{ijot}) will become equal to 1, meaning the operator is being utilized by job j during period t.

Constraint 4.21 states that in any given period, the total used and idled capacity of a machine by all jobs cannot exceed the machine's capacity during the period (R_{mt}). The LHS of the equation includes (1) the total used capacity of the machine by all jobs ($\sum_{i\in I}\sum_{j\in J_i} QS_{ijmt}$) and (2) the total idled capacity of the machine by all jobs ($\sum_{i\in I}\sum_{j\in J_i} QQS_{ijmt}$).

Constraint 4.22 states that in any given period, the total used capacity of an operator by all jobs cannot exceed the operator's capacity during the period (V_{ot}). The summation takes into account the operator requirement of the jobs, Opu_{ijo} (which may be smaller or equal to 1).

4.2 Preprocessing Phase

The speed and performance of an ILP model can be improved by eliminating invalid ranges of variables to reduce the search area of the model. This elimination is done before solving the model in the preprocessing phase.

l_{ij} is the earliest completion time of job j of project i considering only the precedence relations between jobs. If job j of project i has no predecessors, l_{ij} is its earliest possible start time plus its processing time. If job k is the predecessor of job j, l_{ij} is the maximum of (1) its earliest possible start time plus its processing time, and (2) its processing time plus the earliest completion time of the first batch of its predecessors. The complete

code for calculation of l_{ij} can be found in Appendices (section 5.5.1). The following is the Pseudo code for l_{ij} calculation:

- For all jobs of project i:
  - If job has no precedence requirement, set the earliest completion time of job j equal to its earliest possible start time plus its processing time, $(A_{ij} + D_{ij})$.
  - If job k is the predecessor of job j, set the earliest completion time of the job to the maximum of:
    - Its earliest possible start time plus its processing time, $(A_{ij} + D_{ij})$.
    - Its processing time, (D_{ij}) plus earliest completion time of the first batch of its predecessor, $(l_{ik} - D_{ik} + (D_{ik}/B_{ik}))$.

u_{ij} is the latest completion time of job j of project i considering only the precedence relations between jobs. If job j of project i has no successors, u_{ij} is the absolute due-date of project i. If job j is the successor of job k, u_{ik} is the minimum of (1) project i absolute due-date, and (2) the latest completion time of its successor minus the processing time of the last batch of the successor job. The complete code for calculation of u_{ij} can be found in Appendices (section 5.5.1).The following is the Pseudo code for u_{ij} calculation:

- For all jobs of project i:
  - If job has no precedence requirement, set the latest completion time of job j equal to an absolute due-date of project i.
  - If job j is the successor of job k, change the latest completion time of the job to the minimum of:
    - The absolute due-date of project i, (ADD_i).
    - Latest completion time of its successor minus the processing time of the last batch of the successor job $(u_{ij} - D_{ij}/B_{ij})$.

e_i, earliest project completion time is the maximum among the earliest completion time of all jobs of project i.

l_{ij}, u_{ij}, and e_i are used to reduce the search area of the introduced ILP model. Instances of the application can be seen in equation 4.7 as well as the definition of T_i in section 4.1.1.

4.3 Implementation

This model can be used at two different hierarchical production planning levels, namely for high level planning and detailed scheduling by changing the time-scale of the model and passing some constraints of the high level planning solution to the lower level detailed scheduling problem. The higher level planning of the model is aimed at upper management in the job-shop who can use it for aggregate purposes such as resource allocation, customer order promising, due-date planning, and material procurement. On the other hand, the detailed scheduling is targeted towards production managers and is used to develop detailed short-term operation schedules.

In the high level planning phase, the objective is to minimize the total weighted tardiness and the weighted throughput time. In the detailed scheduling phase, the objective is to complete all planned jobs within the specified horizon of the detailed scheduling phase, respecting the results of the high level planning phase. The scheduling will use the original time-scale (usually hours) to generate the flow of jobs and allocate the resources. Resource allocation in the detailed scheduling phase may be different from the high level planning phase as far as the passed constraints of the high level planning are respected. The constraints that are passed from the high level planning phase are:

- The latest completion time of every job according to the high level planning that has been planned to be completed during the detailed scheduling horizon, shall be set equal to the end of the detailed scheduling horizon.

- The latest completion time of the last job of a project according to the high level planning that has been planned to be completed during the detailed scheduling horizon, shall be set equal to its completion time. This is considered to ensure that the rescheduling does not impact the objective function of the high level planning negatively.

- If a job is partially processed during the detailed scheduling horizon, the resources allocated by the high level planning phase will be restrictedly used for processing the job during the detailed scheduling horizon.

- If only one of the setup or its processing job are planned to be processed during the detailed scheduling horizon, the same machine assigned by the high level planning should be restrictedly used for processing the jobs during the detailed scheduling horizon.

If the scheduled jobs could not be performed as scheduled due to any reasons, the data can be updated and the jobs can be re-planned/rescheduled to achieve a new optimum plan. The updating consists of replacing processing times of the jobs accordingly. This requires to replace the processing time of completed jobs with 0, and that of partially processed jobs with estimated remaining processing time. Some examples of reasons to deviate from the schedule are:

- jobs may take more or less time than originally estimated.

- resources may become unavailable.

- material may arrive late.

Figure 4.2 shows an example where jobs are planned to be processed in the current scheduling horizons. Since job 4 of project 8 (8-4) will not be completed in the current scheduling horizon, its remaining process time should be updated to 6 and its completion time must be set to 46. This ensures that the job will complete without any preemption. Also, the same machine performed the job in the current scheduling horizon must be assigned to process the remaining process in the next scheduling horizon.

The constraints of the re-planning phase are:

- If a job is partially processed, the resources used to process the job will be restrictedly used for processing the job during the re-planning phase.

- If only the setup job is processed (partially or completely), the same machine assigned to the setup job will be restrictedly used for its processing jobs during the re-planning phase.

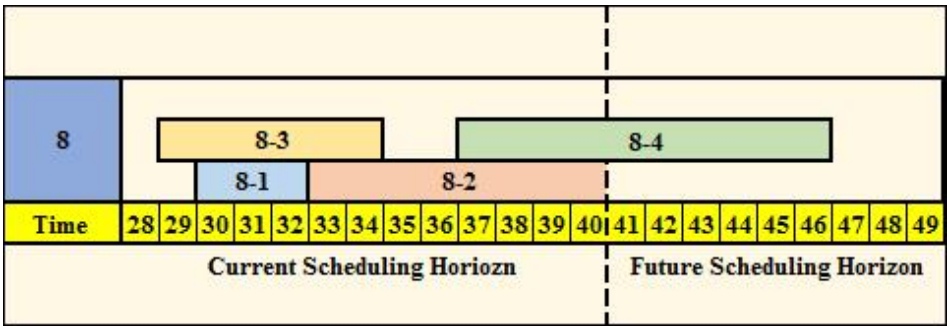


Figure 4.2: An example of overlapping jobs between two scheduling horizon

The following figure shows the cycle or process review for the two phases of high level planning and detailed scheduling. In high level planning phase, more jobs are considered for a long horizon broken into daily time buckets. On the other hand, the detailed scheduling phase considers fewer jobs for a shorter horizon broken into hourly time buckets.

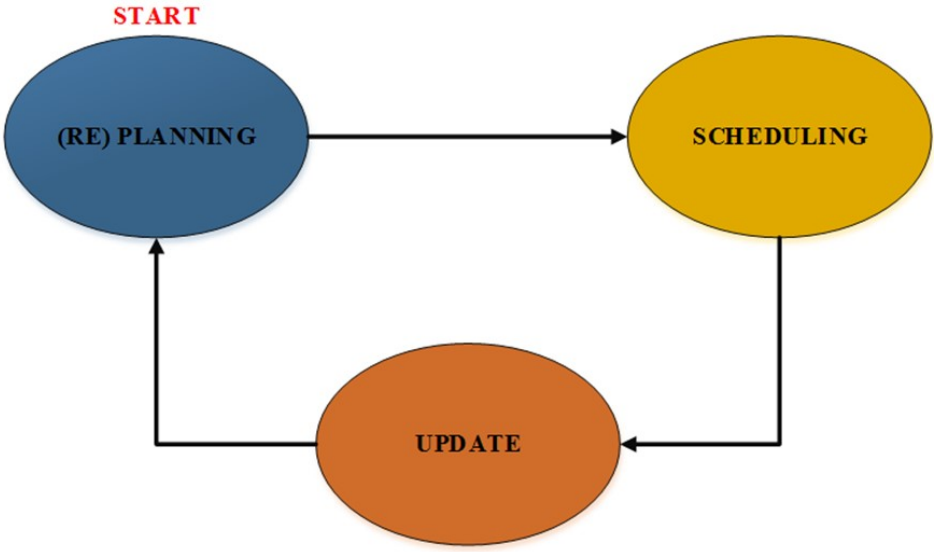


Figure 4.3: ProcessFlow-Planning/Scheduling/Updating/Replanning

The model is coded in Pulp, an open source linear/integer programming modeler written in Python, which can call a variety of solvers (both open source and commercial). The python code for the ILP model can be found in Appendices (section 5.5.1).

Chapter 5

Results and Discussion

In this section, an example is considered to validate, verify and explore the capabilities of the model. The example includes instances of all assumptions and requirements of the model. It consists of 10 projects with total of 36 jobs. The processing times of the jobs is shown in Table 5.1. We have constructed the example using three different schemes of precedence relationships. Any project in the example will have the same precedence relations as shown in Figure 5.1, according to its number of jobs. The odd numbered jobs are the setup for the even numbered jobs. For example, a project with 4 jobs will have identical precedence relations to Figure 5.1 (B), and jobs 1 and 3 are the setup for jobs 2 and 4 respectively. The processing jobs may be carried out in batches. The maximum number of batches for the jobs are shown in Table 5.2.

Table 5.1: Processing time of the jobs (hours)

Projects	Jobs					
	1	2	3	4	5	6
1	3	10	7	10		
2	4	10				
3	1	5				
4	5	10	3	10	4	10
5	4	9	6	10	7	10
6	3	10				
7	3	11				
8	3	8	6	10		
9	7	10	5	11		
10	5	7	7	10		

Six operators and six machines with limited capacity are considered. Operators have various skills, and machines capabilities to process the jobs are different. Consequently, some jobs have a set of capable machines and a set of capable operators to process them. Only one of the resources in each set of machines and operators is needed to perform a job. Table 5.4 shows the set of capable machines to perform each job.

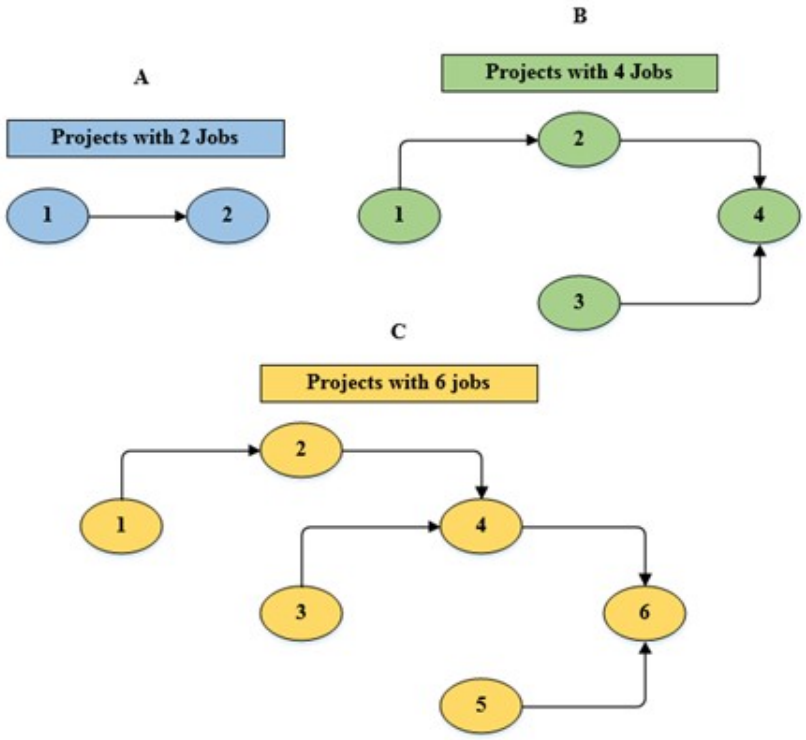


Figure 5.1: Precedence relations for projects with 2, 4 and 6 jobs

Table 5.2: Maximum number of batches of the processing jobs

Projects	Jobs					
	1	2	3	4	5	6
1	Not applicable	5	Not applicable	5	Not applicable	
2		5				
3		1				
4		1		5		10
5		3		5		10
6		5				
7		1				
8		4		5		
9		5		5		
10		2		5		

All the machines are available all the times except machine 4 which is unavailable during day 2 and 3. Table 5.5 shows the set of capable operators to perform each job. The coefficient multiplied by operators represent the partial operator requirement. For example, operator 3 is partially used for all the jobs ($Opu_{ijo} = 0.5$) meaning that he may be able to handle more than one job at any given period. All the operators are available all the times except operator 2 which is unavailable during day 4 and 5. Desired and absolute due-dates are known for each project. No cost is incurred if a project is completed before its desired due-date. If it is completed after the desired due-date, a lateness cost is imposed. The lateness cost depends on the priority of the project and the lateness. All the projects must be completed before their absolute due-dates. Desired and absolute due-dates, and priority of the project are shown in Table 5.3.

Table 5.3: A hypothetical example with 10 projects

Project	Number of Jobs	Arrival Day	Desired Due-date	Absolue Due-Date	Lateness Cost /unit of time
1	4	0	5	7	200
2	2	0	3	7	200
3	2	0	2	7	200
4	6	0	5	12	200
5	6	0	5	7	200
6	2	1	4	7	100
7	2	1	4	7	100
8	4	1	6	12	100
9	4	1	7	12	100
10	4	1	6	12	100

Table 5.4: Machine requirements of the jobs

Project	Job					
	1	2	3	4	5	6
1	3,4,5,6	3,4,5,6	2,3,4	2,3,4		
2	1,3	1,3				
3	1,2	1,2				
4	1,5,6	1,5,6	1,2,3,4	1,2,3,4	2,3	2,3
5	1,3,4,6	1,3,4,6	2,5,6	2,5,6	2,5,6	2,5,6
6	1,4,5	1,4,5				
7	1,2,3	1,2,3				
8	4,6	4,6	2,3,4	2,3,4		
9	1,4	1,4	1,2,3,4	1,2,3,4		
10	5	5	4,5	4,5		

Table 5.5: Operator requirements of the jobs

Project	Job					
	1	2	3	4	5	6
1	1,4,5,6	1,4,5,6	(0.5)*3,4	(0.5)*3, 4		
2	1	1				
3	1,2, (0.5)*3	1,2, (0.5)*3				
4	(0.5)*3,5,6	(0.5)*3, 5,6	2	2	2, (0.5)*3	2, (0.5)*3
5	1,2, (0.5)*3,4,6	1,2, (0.5)*3,4,6	1,(0.5)*3,4,5,6	1,(0.5)*3,4,5,6	1,4	1,4
6	2,4	2,4				
7	1,(0.5)*3,5	1,(0.5)*3,5				
8	1,2,5	1,2,5	(0.5)*3,4,5	(0.5)*3,4,5		
9	5,6	5,6	1,(0.5)*3,4	1,(0.5)*3,4		
10	3,4,5	3,4,5	2,4	2,4		

5.1 ILP Solution

The introduced example is solved by the proposed ILP model with the minimization of total weighted tardiness and the weighted throughput time as the objective functions. The details of the solution is presented in following. For the above example, it takes 96 seconds for the model to find the optimum solution and output the results on a personal computer with Intel core i5, 3.2 GHz processor, installed memory (RAM) 8 GB, and Python version 2.7. In Figure 5.2, the Gantt Chart represents the start and completion time of the jobs in the example solved to obtain the optimum solution for the high level planning phase. All the jobs are planned to be completed in 10 days. All the example's assumptions and requirements as explained earlier are respected when obtaining the optimum solution.

Table 5.6 shows the start and completion days as well as the penalty cost of the projects related to the optimum solution of the high level planning phase. The planning problem has 2396 variables. The solution's objective function is 1500 which is reflected in 5 late projects; Project 4, 6, 7, 9 and 10.

We opt to run the detailed scheduling phase for the next 5 days (40 hours). All completed or partially processed jobs in the first 5 days of the high level planning horizon are considered to be scheduled in the detailed scheduling horizon. Therefore, all the jobs of project 1, 2, 3, 5, 6, 7, and 8 need to be considered. In figure 5.3, the Gantt Chart shows the start and completion time of the jobs planned in the detailed scheduling

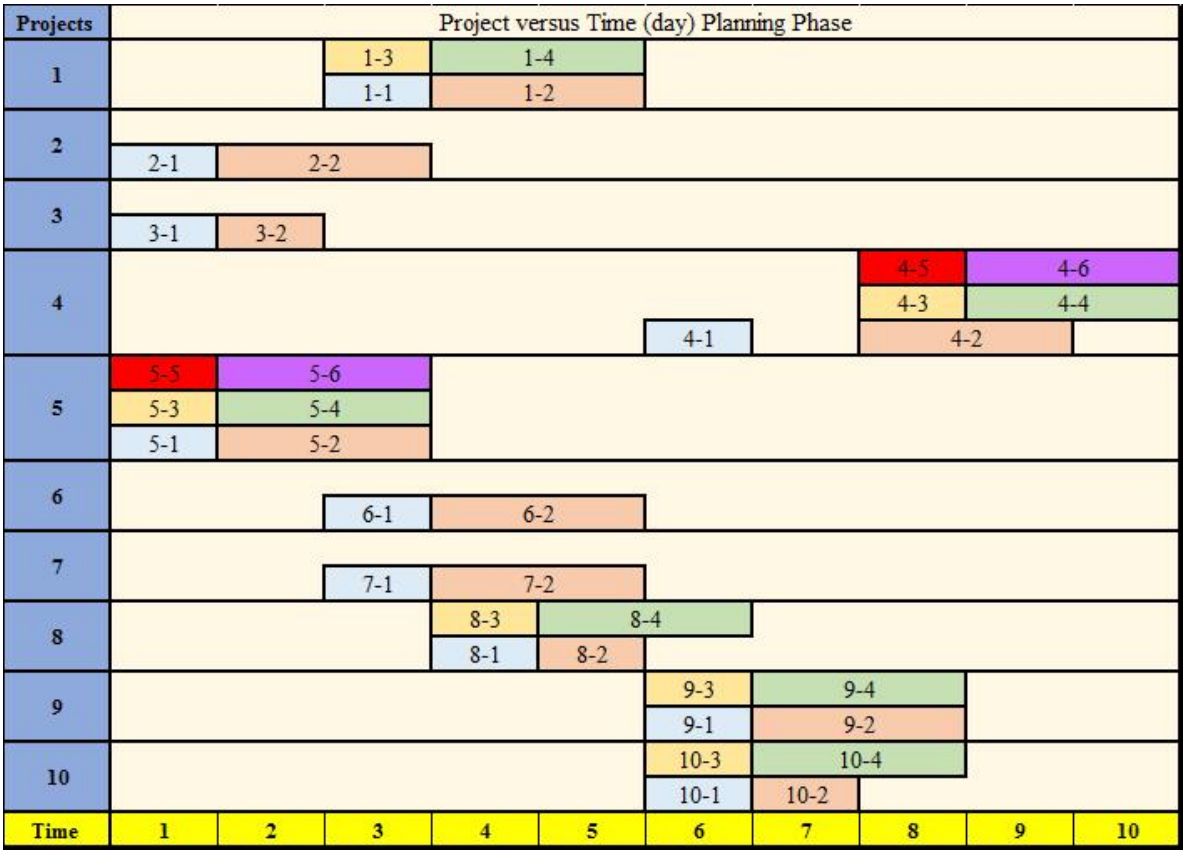


Figure 5.2: Gantt Chart- (Projects versus Time) for high level planning phase

Table 5.6: Result of the high level planning phase

Project	Start Day	Completion Day	Lateness Cost
1	0	5	0
2	0	3	0
3	0	2	0
4	4	10	1000
5	0	3	0
6	2	5	100
7	4	5	100
8	2	6	0
9	6	8	100
10	4	8	200

phase.

Projects 1, 5, and 8 are taking advantage of batching to reduce the completion time of the projects (e.g. project 1: job 2 and 4, project 3: job 2, 4 and 6). Batching is not applied to setup jobs as shown in the Gantt Chart (e.g. project 1: job 1 and 2 or job 3 and 4). The result shows that the selected jobs for the detailed scheduling horizon are planned to be processed within 37 hours.

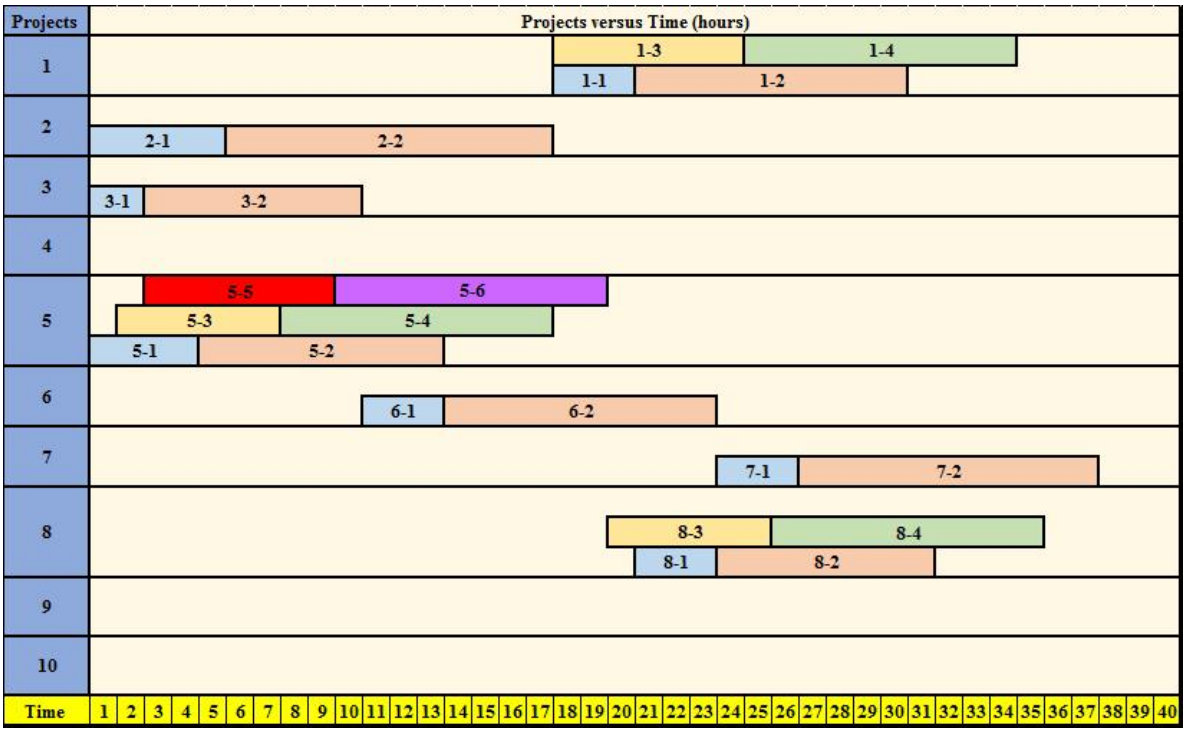


Figure 5.3: Gantt Chart- (Projects versus Time) for the detailed scheduling phase

The scheduling problem has 6374 variables. Therefore, for this case, the planning problem has less variables than the scheduling problem.

According to the high level planning solution (5.2), the completion time for job 4 of project 8 is out of scheduling horizon. If due to any reason the job could not be completed in the detailed scheduling horizon or progressed more than expected, remaining jobs must be re-planned/re-scheduled.

Figure 5.4 shows the Gantt Chart of the machines utilization. Setups and their corresponding processing jobs use the same machine (e.g. project 3: job 1 and 2, project 2:

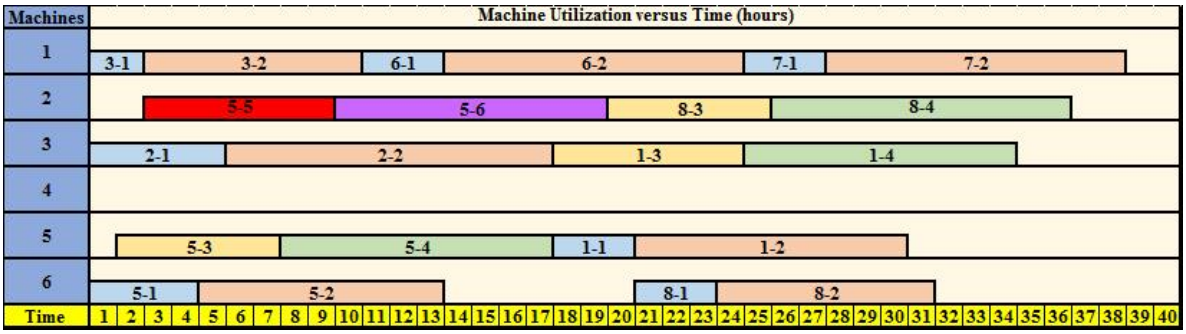


Figure 5.4: Gantt Chart- Machine Utilization versus Time for the detailed scheduling phase

job 1 and 2). Table 5.4 is used to show that machines are allocated to the jobs respecting their machining requirement. Machine 4 is not allocated to any job between time 16 and 32 due to its unavailability. Since the operator required for the setup may be different from the operator required for the processing job, it is sometimes advantageous to finish the setup job when the setup operator is available, even though processing job itself might start later, which introduces idling for the machines. The optimum solution of this example does not recommend any idling between the setup and processing jobs.

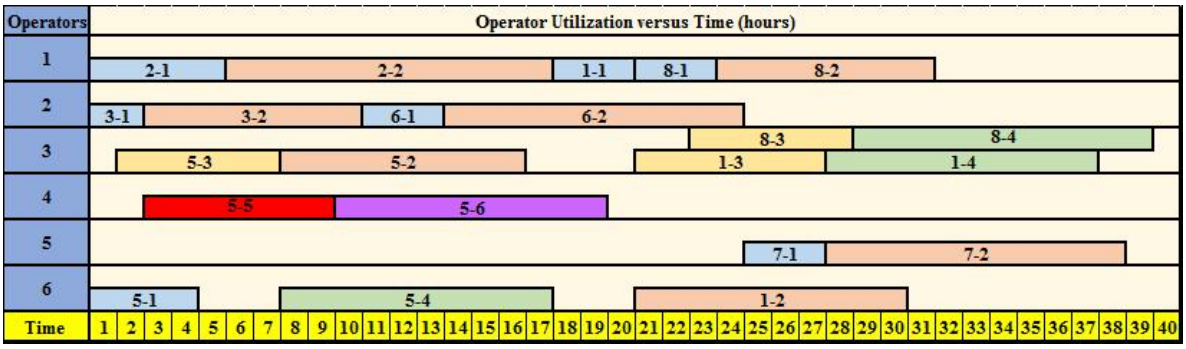


Figure 5.5: Gantt Chart- Operator Utilization versus Time for scheduling phase

Figure 5.5 shows the Gantt Chart of operators utilization respecting the set of capable to process each job (Table 5.5).

It also shows, operator 3 is allocated to process two jobs between times 23 and 37. Operator 2 is not assigned to any jobs due to its unavailability between times 24 to 40.

The objective of the detailed scheduling phase is to complete all planned jobs within the

Table 5.7: Result of the first detailed scheduling phase

Scheduling - Phase 1			
Project	**Number of Jobs**	**Start Time**	**Completion Time**
1	4	18	34
2	2	0	17
3	2	0	10
4	0		
5	6	0	19
6	2	11	23
7	2	24	37
8	4	20	35
9	0		
10	0		

specified horizon of the detailed scheduling phase (5 days). For the detailed scheduling phase, it takes 36 seconds for the model to solve the problem. Table 5.7 shows the solution of the detailed scheduling phase. The completion times of the all the jobs is within the 40 hours (the detailed scheduling horizon).

In a real world situation, a detailed scheduling will rarely be performed exactly as planned. For that reason, a third phase is required to update the progress of the projects before re-running the high level planning and detailed scheduling phases. In what follows, the details of this phase (updating phase) is explained through an example.

Assume that when implementing scheduling as detailed in Figure 5.3, job 4 of project 1 is only 70% done despite it being expected to finish within the detailed scheduling horizon. Also assume that job 4 of project 8 that was planned to be 20% done is 60% done. In order to get the optimum plan/schedule after the discrepancies, it is required to re-run the planing and scheduling phase after updating the data as explained in section 4.3. It also requires the constraints discussed in that section to be passed to the re-planning and re-scheduling phases.

The earlier example is also solved with several desired due-dates for the projects and throughput coefficients. A throughput coefficient is a constant number multiplied by the completion time in the objective function to decrease or increase the effect of the throughput on the solution. Due-dates are given in the following types: (1) Flexible due-dates: almost all projects can be completed on-time. All the projects are given

due-dates greater than their earliest completion time (e_i), (2) Reasonable due-dates: sufficient time is given so most projects to be completed on or before the due-dates. The due-dates are greater than e_i for almost 50"%" of the projects, (3) Tight due-dates: inadequate time is given so that most projects will finish late. About 25"%" of the projects are given due-dates greater than their e_i, and (4) Very Tight due-dates: almost no projects can be completed on or before the due-dates. The due-dates are either equal or less than their e_i. The more restricted the due-dates are, the more resources are required to complete the jobs on or before the due-dates. On the other hand, if the due-dates are flexible, less resources are required to complete the jobs on-time. Examples with various desired due-dates for the projects are solved with the following throughput coefficients; 0.01, 0.1, 1, and 10. The results are shown in Table 5.8. The test is considered to measure the impact of the desired due-dates and throughput coefficient on the computational time.

Table 5.8: Results from the example with different due-date types and throughpout coefficients

Throughput Coefficient	Comparison Factors	Reasonable	Tight	Very Tight	Flexible
0.01	Objective Value	1815	3515	4715	1115
	% Missed Due-dates	0.6	0.8	0.9	0.4
	Time (sec)	96.65	161.73	102.44	839.17
0.1	Objective Value	1950	3650	4850	1250
	% Missed Due-dates	0.6	0.8	0.9	0.4
	Time (sec)	91.94	171.29	105.43	746.08
1	Objective Value	3300	5000	6200	2600
	% Missed Due-dates	0.6	0.8	0.9	0.4
	Time (sec)	109.26	263.14	87.52	824.15
10	Objective Value	16800	18500	19700	16100
	% Missed Due-dates	0.6	0.9	0.9	0.4
	Time (sec)	98.05	182.42	139.48	701.92

In Table 5.8, the impact of the due-dates and throughput coefficient on computation time and the average weighted tardiness and throughput time are not huge but non-negligible. When the due-dates are given to the projects reasonably, the model obtains a solution quickly in most cases. Tight due-dates makes the problem more complex which takes longer for the model to get the solution with the minimum objective value. In case of very tight due-dates, the model finds solutions much faster than the tight due-dates

since the search space is smaller. Flexible due-dates are very time-consuming because the search space is bigger. In most cases, in the given example, different due-date types have some impact on the computation time while the variation of throughput coefficient shows less impact. The computation times in flexible due-dates are considerably higher among other due-date types. In Table 5.8, the times in red represent the minimum computation time for each coefficient. Examples with reasonable due-dates mostly obtained results faster than others but, the differences are not very big. In the example with very tight due-dates, and 1 as the coefficient, the solution was obtained faster than the other due-date types. In general, the coefficient does not seem to change the time and the variation looks very random. As a result, the order in all of the computation times seems to be (1) Reasonable, (2) Very tight, (3) Tight, and (4) Flexible except for the case where the coefficient is 1. This conclusion is based on one example only and is not statistically proven and more research needs to be done on this.

5.2 Heuristic Approaches

In this section we intend to compare the performance of the introduced ILP model with some well-known heuristic models. To do so, some examples with different project sizes are solved by the models, and the weighted tardiness, makespan, and average project lateness are compared. Batching is not considered in the heuristic models, which handicaps them to a certain extent.

SPT, EDD, SPT-EDD and EDD-SPT are the heuristic approaches tested and compared with the ILP method. The SPT sequences the jobs in increasing order of their processing times. Whenever a resource is available, the shortest job ready in the queue at the time, will begin processing. The queue consists of jobs with all their predecessors completed. The Pseudo code of the SPT model is as follows:

- Set time to t=0
- Generate a list (All-Jobs) of all remaining jobs of projects.
- From the generated list, select the jobs that either has no predecessors or their predecessors are completed. Generate a new list (Jobs at time t) from the selected

jobs.

- Sort all the jobs of the new list (Jobs at time t) in ascending order based on their processing time.
- Start from the first job in the new list (Jobs at time t) and check if its machine and operator requirements are available during its processing time.
  - If yes, assign the corresponding machine and operator to the job and update resources availability during the given periods.
  - If no, move to the next job in the list (Jobs at time t).
- Carry out the same procedure for all the remaining jobs in the list (Jobs at time t).
- Update the All-Jobs list according to the remaining jobs in the list (Jobs at time t).
- Set time to t+1
- Continue until all the jobs in All-Jobs list are scheduled.

All the steps of the other heuristic models are identical to SPT except for sorting the new list (item 4). The EDD sorts the jobs in increasing order of their due-dates. The SPT-EDD first sorts the jobs in increasing order of their processing times and then sort them in increasing order of their due-dates. The EDD-SPT first sorts the jobs in increasing order of their due-dates and then sort them in increasing order of their processing time. The Pseudo code of EDD, SPT-EDD, EDD-SPT approaches are similar to SPT. The only difference is that jobs in EDD, SPT-EDD, EDD-SPT jobs are sorted in ascending order of their due-dates, processing times and due-dates, due-dates and processing time, respectively. The python code for all the heuristic model can be found in Appendices (SPT: 5.5.1, EDD: 5.5.1, SPT-EDD: 5.5.1, EDDSPT: 5.5.1).

The example of the previous section is also solved using the heuristic models and the results are presented in table 5.9. Not surprisingly, none of the solutions obtained by heuristics approaches is optimal. It must be noted that the heuristics solutions

are disadvantaged because they cannot take advantage of batching. The ILP model obtained the global optimum solution and with advances in computation power, the ILP solution can be obtained in a reasonable time. In the following sections, more complex examples are solved to assess the model performance.

Table 5.9: Comparison of the result from ILP model with heuristic approaches

Approach	Average Project Lateness (Day)	Makespan (days)	Computation Time (Sec)	Weighted Tardiness	Distance From Optimal
ILP	1.0	10	96	1500	
SPT	4.5	15	5	7100	373%
EDD	2.8	12	5	4200	180%
SPT-EDD	3.9	13	5	6100	307%
EDD-SPT	2.8	13	5	4500	200%

5.3 Examples with other problem sizes

More examples with a different number of projects (10, 20, 30, 40, and 50) are considered in this section. The purpose is to test the model performance with different problem sizes. The results from ILP model and heuristic approaches are compared and shown in the following figures. All examples include instances of all assumptions and requirements of the model. 10 operators and 10 machines are considered to perform the jobs. These examples are considered to show the effect of problem size on the solution and computation times (all runs were made on a personal computer with Intel core i5, 3.2 GHz processor, installed memory (RAM) 8 GB, and Python version 2.7.). The list of all 50 projects in these examples are shown in Table 5.10. The first 10, 20, 30, 40, and 50 projects are considered for example 1, 2, 3, 4, and 5, respectively.

The results shows that an increase in problem size increases the network complexity of the problem so that it takes the ILP model longer to obtain an optimum solution. However, the computation time of heuristic approaches for all the examples is short and consistent, but the objective functions are far from optimum. The computation times are shown in Table 5.11. Once again, all runs were made on a personal computer with Intel core i5, 3.2 GHz processor, installed memory (RAM) 8 GB, and Python version 2.7.

Table 5.10: The examples with 10,20,30,40, and 50 projects

Project	Number of Jobs	Arrival Day	Desired Due-date	Penalty Cost /unit of time
1	4	0	40	200
2	2	0	40	200
3	2	0	24	200
4	6	0	40	200
5	6	0	40	200
6	2	8	48	100
7	2	8	48	100
8	4	8	48	100
9	4	8	48	100
10	4	8	40	100
11	4	24	56	90
12	2	24	48	90
13	2	24	48	90
14	6	24	56	90
15	6	24	64	90
16	2	32	48	90
17	2	32	56	90
18	4	32	56	90
19	4	32	56	90
20	4	32	72	90
21	4	48	80	80
22	2	48	72	80
23	2	48	56	80
24	6	48	80	80
25	6	48	88	80
26	2	56	72	80
27	2	56	72	80
28	4	56	80	80
29	4	56	96	80
30	4	56	96	80
31	4	64	96	70
32	2	64	88	70
33	2	64	88	70
34	6	64	96	70
35	6	64	96	70
36	2	72	88	70
37	2	72	88	70
38	4	72	104	70
39	4	72	104	70
40	4	72	104	70
41	4	80	120	60
42	2	80	128	60
43	2	80	128	60
44	6	80	128	60
45	6	80	128	60
46	2	88	128	60
47	2	88	128	60
48	4	88	128	60
49	4	88	128	60
50	4	88	128	60

ILP Model	
Project	Computation time (min)
10	4.9
20	6.7
30	32.3
40	112.3
50	116.8

Table 5.11: Computation time using ILP model for solving problems with various sizes

The difference between the optimal ILP solution and the heuristic solutions is not huge for smaller problems but it becomes larger as problem size increases. As earlier, it must be noted that the heuristics solutions are disadvantaged because they cannot take advantage of batching.

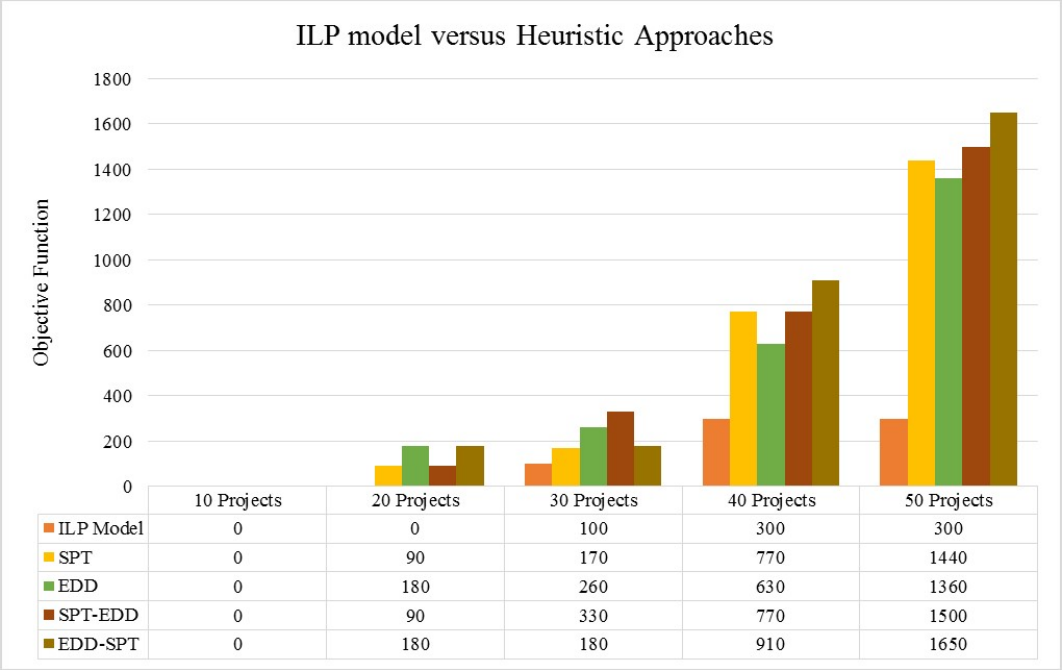
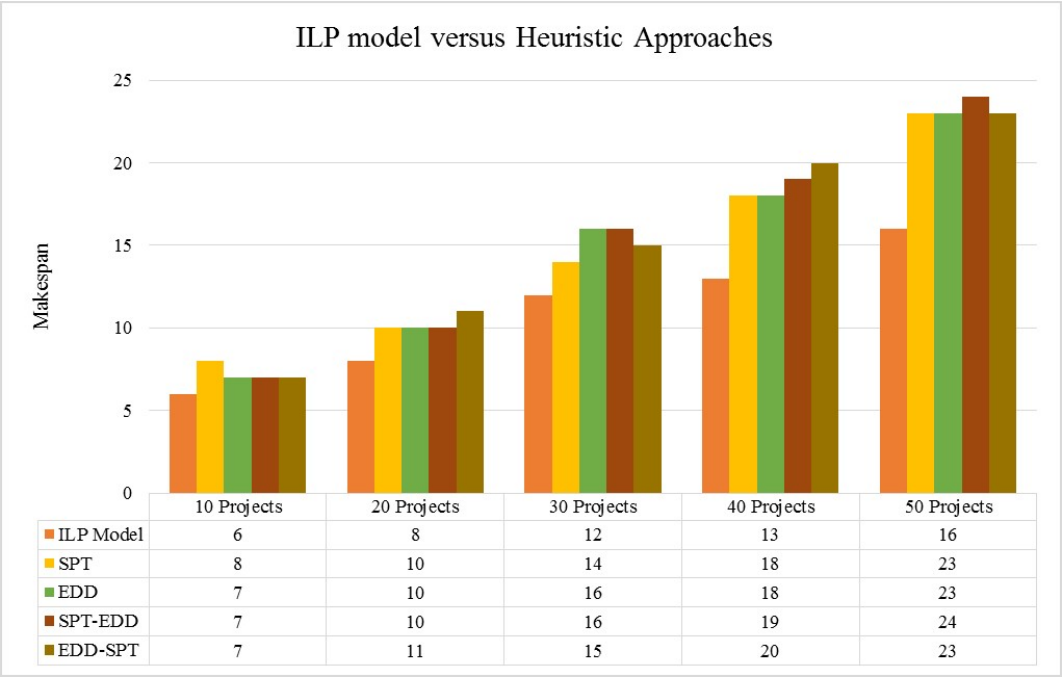


Figure 5.6: ILP model versus Heuristic Approaches - Makespan comparison for various problem sizes



	10 Projects	20 Projects	30 Projects	40 Projects	50 Projects
ILP Model	6	8	12	13	16
SPT	8	10	14	18	23
EDD	7	10	16	18	23
SPT-EDD	7	10	16	19	24
EDD-SPT	7	11	15	20	23

Figure 5.7: ILP model versus Heuristic Approaches - Objective Value comparison for various problem's size

5.4 Comparison between ILP Model solution and Company's Performance

In this section, the company performance is compared with the ILP model and heuristic models solution. In this comparison, the total weighted tardiness of the projects is not a suitable factor for comparing the actual performance with other approaches. The due-dates given to the projects are not the original due-dates of the projects. In practice, the company changes the due-dates regularly when current due-dates cannot be met. There is not an accessible record of actual due-dates available. Hence, the time to complete all the projects (makespan) is considered to compare the results.

In this real-life example, 50 projects consisting of 294 jobs are considered. The analysis starts from 09/01/2015 (the earliest arrival date of the projects) and ends by 11/05/2015 (latest completion date of the projects). In Table 5.12, the start date is when a project arrives at the shop, and the end date is when the project is completed. *Actual performance* represents the total number of days the projects stayed in the shop, excluding weekends. Column of *ILP model (Duration)* shows the duration of the projects obtained

Table 5.12: The comparison of the completion times of the projects performed by the company and the proposed plan by ILP model

Project	Start	End	Actual Performance	ILP Model (Duration)	Improved
1	09/01/2015	24/02/2015	35	3	YES
2	09/01/2015	24/02/2015	35	3	YES
3	09/01/2015	17/03/2015	35	3	YES
4	09/01/2015	17/03/2015	50	13	YES
5	09/01/2015	25/02/2015	50	4	YES
6	09/01/2015	25/02/2015	36	3	YES
7	09/01/2015	25/02/2015	36	3	YES
8	09/01/2015	25/02/2015	36	4	YES
9	09/01/2015	24/02/2015	36	4	YES
10	09/01/2015	23/03/2015	35	9	YES
11	09/01/2015	24/02/2015	54	5	YES
12	09/01/2015	23/03/2015	35	6	YES
13	09/01/2015	02/03/2015	54	6	YES
14	09/01/2015	02/03/2015	39	10	YES
15	09/01/2015	17/03/2015	39	5	YES
16	09/01/2015	24/02/2015	50	5	YES
17	09/01/2015	25/02/2015	35	8	YES
18	09/01/2015	25/02/2015	36	7	YES
19	09/01/2015	26/03/2015	36	5	YES
20	09/01/2015	02/03/2015	57	9	YES
21	09/01/2015	02/03/2015	39	11	YES
22	09/01/2015	17/03/2015	39	6	YES
23	09/01/2015	10/03/2015	50	8	YES
24	09/01/2015	24/02/2015	45	16	YES
25	09/01/2015	24/02/2015	35	15	YES
26	09/01/2015	10/03/2015	35	17	YES
27	09/01/2015	17/03/2015	45	10	YES
28	09/01/2015	23/03/2015	50	11	YES
29	09/01/2015	24/02/2015	54	10	YES
30	09/01/2015	25/02/2015	35	10	YES
31	09/01/2015	25/02/2015	36	9	YES
32	09/01/2015	30/03/2015	36	9	YES
33	09/01/2015	23/03/2015	59	11	YES
34	13/01/2015	02/02/2015	54	10	YES
35	14/01/2015	17/02/2015	17	10	YES
36	15/01/2015	16/04/2015	27	12	YES
37	15/01/2015	14/04/2015	66	12	YES
38	16/01/2015	06/02/2015	66	17	YES
39	19/01/2015	22/01/2015	16	9	YES
40	19/01/2015	22/01/2015	4	11	NO
41	19/01/2015	09/02/2015	4	15	NO
42	19/01/2015	02/02/2015	16	15	YES
43	19/01/2015	09/02/2015	11	40	NO
44	19/01/2015	20/01/2015	16	40	NO
45	20/01/2015	05/05/2015	2	27	NO
46	20/01/2015	28/04/2015	76	40	YES
47	20/01/2015	11/05/2015	71	39	YES
48	20/01/2015	05/05/2015	82	27	YES
49	20/01/2015	05/05/2015	76	26	YES
50	20/01/2015	10/04/2015	76	28	YES
		Makespan	**82**	**40**	**90.0%**

from the model. In column of *Improved*, *YES* means that the company's completion time for the given project was improved by the ILP model while *NO* means the other way around. According to Table 5.12, the obtained solution by the ILP model outperforms the company performance. As a result, 90% of the projects completion time were improved. The makespan given by the model is 40 days while the company completed all the projects in 82 days. However, we anticipate that the 42 days difference is not purely result of ILP planning. The comparison of the completion times of the projects performed by the company and the proposed plan by ILP model can be seen in Table 5.12.

The ILP model is also compared with the heuristics approaches. According to Table 5.13, ILP model outperforms the heuristic approaches. Among the heuristic model, EDD obtained smaller makespan which is consistent with the previous conclusion. On the other hand, EDD-SPT obtained the minimum tardiness. Both makespan and tardiness obtained by heuristic approaches are far from the optimal solution.

Table 5.13: Comparison of ILP model with heuristic approaches using the real-life example

	SPT	EDD	SPT-EDD	EDD-SPT	Optimization
Makespan (days)	76	47	65	54	40
Tardiness	161	105	132	98	47.5

5.5 Conclusion and Future Research

Based on a real-life case study at a machine shop, we considered some extensions to multiple resource constraint JSSP with generalized precedence relations and proposed a linear integer programming model to solve the problem. The extensions are project priority, alternative resources, partial resource usage, and batching. According to the literature review, among the extensions, partial resource usage and batching are the new contributions to job-shop scheduling and also the combination of all these extension has not been considered in any single research. Project priority determines competing projects, priority to access the limited resources respect to their desired due-dare. With

alternative resources, more than one resource may exis with the required capabilities to perform a job. In partial resource usage jobs may not require the resource's full capacity to perform them. Hence, operators may be assigned to more than one job during a particular period, to the extent of their full capacity. Batching breaks down a job into smaller jobs to allow the successors start processing earlier and thus improving the efficien while preventing any preemption.

The introduced ILP model presents a planning/scheduling framework. In the planning stage, the model produces a long-term plan for the production. Then, the constraints from the high level planning phase is passed to the detailed scheduling phase to provide a short-term schedule for the production. The objective is to minimize the total weighted tardiness and the weighted throughput time for all the projects. The model is coded in Pulp, a linear/integer programming modeler written in Python, which can call different solvers to solve the problem.

The proposed model is compared with multiple heuristics (SPT, EDD, SPT-EDD, and EDD-SPT) models as well as the actual performance at the shop. The heuristic approaches find relatively good solutions fast, but they are far from the optimality. The bigger the size of the problem, the farther the solution from the optimality. Among the heuristic approaches, EDD outperforms others with the minimum total weighted tardiness and makespan. According to the *ILP vs. Heuristics* table, ILP model outperforms the heuristic models by 106.31% (tardiness) comparing to the next best solution which is the EDD-SPT.

In a real-life example, the ILP model obtained solution where 90% of the projects had an smaller completion time. The model completed all the projects in 40 days while the company's makespan was 82 days. The Same example was solved using heuristic approaches. All heuristic approaches performed better than the company's performance. Among the heuristic approaches, EDD obtained the minimum makespan while EDD-SPT got the lowest tardiness. Nonetheless, the model proved to have a better solution comparing to the heuristic approaches and the actual performance of the company.

5.5.1 Future Research

- Resource Efficiency: Operator skills and machine specification are not considered in this thesis. The model assume that alternative operators and machines to perform the jobs by a constant processing time. However, operator's with various skills might be able to perform jobs more or less efficient. On the other hand, different machine types might have different efficiencies. The consideration of resource efficiency can be a future extension of the model.

- Consideration of Splitting: The concept of splitting is similar to batching. however, splitting considers breaking down jobs into batches potentially performed by different resources and allows for preemption. Splitting could be considered in the model as a future research.

- Flexibility of Resource Allocation: The considered resources in the model are operators and machines. More resources (e.g. tools) may be required to perform the jobs. The model can be adopted to consider additional resources in the problems.

- This research only investigated the impact of one throughput coefficient on one example. To better study the impact of various throughput coefficients on model's performance further studies is required.

Bibliography

[1] Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management science*, 34(3):391–401, 1988.

[2] Behrouz Afshar-Nadjafi, Hamid Karimi, Amir Rahimi, and Somayeh Khalili. Project scheduling with limited resources using an efficient differential evolution algorithm. *Journal of King Saud University-Engineering Sciences*, 2013.

[3] Francisco Ballestín, Vicente Valls, and Sacramento Quintanilla. Pre-emption in resource-constrained project scheduling. *European Journal of Operational Research*, 189(3):1136–1152, 2008.

[4] Umut Beşikci, Ümit Bilge, and Gündüz Ulusoy. Resource dedication problem in a multi-project environment. *Flexible Services and Manufacturing Journal*, 25(1-2):206–229, 2013.

[5] Lucio Bianco and Massimiliano Caramia. A new formulation for the project scheduling problem under limited resources. *Flexible Services and Manufacturing Journal*, 25(1-2):6–24, 2013.

[6] Fayez F Boctor. Resource-constrained project scheduling by simulated annealing. *International Journal of Production Research*, 34(8):2335–2351, 1996.

[7] Tyson R Browning and Ali A Yassine. A random generator of resource-constrained multi-project scheduling problems. *Urbana*, 51:61801, 2007.

[8] Peter Brucker, Bernd Jurisch, and Bernd Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete applied mathematics*, 49(1):107–127, 1994.

[9] FTS Chan, TC Wong, and LY Chan. Flexible job-shop scheduling problem under resource constraints. *International Journal of Production Research*, 44(11):2071–2089, 2006.

[10] Edward W Davis and James H Patterson. A comparison of heuristic and optimum solutions in resource-constrained project scheduling. *Management science*, 21(8):944–955, 1975.

[11] Bert De Reyck et al. A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research*, 111(1):152–174, 1998.

[12] Bert De Reyck and Willy Herroelen. The multi-mode resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research*, 119(2):538–556, 1999.

[13] Hsiao-Lan Fang, Peter Ross, and David Corne. *A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems.* University of Edinburgh, Department of Artificial Intelligence, 1993.

[14] Guillermo Gallego. Production management. *lecture notes, Columbia University, New York, NY*, 1995.

[15] Sönke Hartmann and Dirk Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010.

[16] Willy Herroelen, Bert De Reyck, and Erik Demeulemeester. Resource-constrained project scheduling: a survey of recent developments. *Computers & Operations Research*, 25(4):279–302, 1998.

[17] Oumar Koné, Christian Artigues, Pierre Lopez, and Marcel Mongeau. Event-based milp models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3–13, 2011.

[18] Yazid Mati. Minimizing the makespan in the non-preemptive job-shop scheduling with limited machine availability. *Computers & Industrial Engineering*, 59(4):537–543, 2010.

[19] Carlos A Méndez, Jaime Cerdá, Ignacio E Grossmann, Iiro Harjunkoski, and Marco Fahl. State-of-the-art review of optimization methods for short-term scheduling of batch processes. *Computers & Chemical Engineering*, 30(6):913–946, 2006.

[20] Jairo R Montoya-Torres, Edgar Gutierrez-Franco, and Carolina Pirachicán-Mayorga. Project scheduling with limited resources using a genetic algorithm. *International Journal of Project Management*, 28(6):619–628, 2010.

[21] A Alan B Pritsker, Lawrence J Waiters, and Philip M Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management science*, 16(1):93–108, 1969.

[22] M Rajkumar, P Asokan, N Anilkumar, and T Page. A grasp algorithm for flexible job-shop scheduling problem with limited resource constraints. *International Journal of Production Research*, 49(8):2409–2423, 2011.

[23] Hai-yan Wang, Yan-wei Zhao, Xin-li Xu, and Wan-Liang Wang. A batch splitting job shop scheduling problem with bounded batch sizes under multiple-resource constraints using genetic algorithm. In *Cybernetics and Intelligent Systems, 2008 IEEE Conference on*, pages 220–225. IEEE, 2008.

[24] Chiu-Chi Wei, Ping-Hung Liu, and Ying-Chin Tsai. Resource-constrained project management using enhanced theory of constraint. *International Journal of Project Management*, 20(7):561–567, 2002.

[25] Lawrence M Wein and Philippe B Chevalier. A broader view of the job-shop scheduling problem. *Management science*, 38(7):1018–1033, 1992.

[26] Wu Ying and Li Bin. Job-shop scheduling using genetic algorithm. In *Systems, Man, and Cybernetics, 1996., IEEE International Conference on*, volume 3, pages 1994–1999. IEEE, 1996.

[27] Liu Yongxian, Liu Xiaotian, and Zhao Jinfu. Research on job-shop scheduling optimization method with limited resources. *The International Journal of Advanced Manufacturing Technology*, 38(3-4):386–392, 2008.

[28] Rui Zhang and Cheng Wu. A double-layered optimisation approach for the integrated due date assignment and scheduling problem. *International Journal of Production Research*, 50(1):5–22, 2012.

[29] N Zribi, I Kacem, A El-Kamel, and P Borne. Minimizing the total tardiness in a flexible job-shop. In *World Automation Congress*. Citeseer, 2006.

Appendix 1 Preprocessing

```
import numpy as np
import xlrd
import time
import datetime
import matplotlib.pyplot as plt
import sys
import shutil
import os
from matplotlib.ticker import MultipleLocator,
   FormatStrFormatter
from datetime import datetime
print '#',datetime.now().strftime('%Y-%m-%d %H:%M:%S'),'#'
book = xlrd.open_workbook("datanew.xls")
sheet = book.sheet_by_index(18)
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
   ,
 sheet.nrows, sheet.ncols)
for i in xrange(sheet.nrows):
for j in xrange(sheet.ncols):
Process = sheet.cell_value(i,j)
#print 'Process', Process
sheet = book.sheet_by_index(0)
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
   ,
sheet.nrows, sheet.ncols)
Project = []
for i in xrange(sheet.nrows):
for j in xrange(sheet.ncols):
Project.append(int(sheet.cell_value(i,j)))
#print 'Project', Project
sheet = book.sheet_by_index(1)
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
   ,
```

```
sheet.nrows, sheet.ncols)
ActualDuration = []
for i in xrange(sheet.nrows):
ActualDuration.append(sheet.row_values(i))
for i in xrange(sheet.nrows):
while '' in ActualDuration[i]:
ActualDuration[i].remove('')
#print 'Actual Duration', ActualDuration
sheet = book.sheet_by_index(11)
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
   ,
 sheet.nrows, sheet.ncols)
InflationRate = []
for i in xrange(sheet.nrows):
InflationRate.append(sheet.row_values(i))
for i in xrange(sheet.nrows):
while '' in InflationRate[i]:
InflationRate[i].remove('')
#print 'InflationRate', InflationRate
#
#
sheet = book.sheet_by_index(4)
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
   ,
 sheet.nrows, sheet.ncols)
AbsoluteDueDate = []
for i in xrange(sheet.nrows):
for j in xrange(sheet.ncols):
AbsoluteDueDate.append(int(sheet.cell_value(i,j)))
#print 'AbsoluteDueDate', AbsoluteDueDate

#
#
sheet = book.sheet_by_index(3)
```

```
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
    ,
 sheet.nrows, sheet.ncols)
DesiredDueDate = []
for i in xrange(sheet.nrows):
for j in xrange(sheet.ncols):
DesiredDueDate.append(int(sheet.cell_value(i,j)))
#print 'DesiredDueDate', DesiredDueDate

#
#
sheet = book.sheet_by_index(10)
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
    ,
 sheet.nrows, sheet.ncols)
Weight = []
for i in xrange(sheet.nrows):
for j in xrange(sheet.ncols):
Weight.append(int(sheet.cell_value(i,j)))
#print 'Weight', Weight

#
#
sheet = book.sheet_by_index(2)
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
    ,
 sheet.nrows, sheet.ncols)
Arrivall =[]
Arrival=[]
for i in xrange(sheet.nrows):
Arrivall.append(sheet.row_values(i))
for i in xrange(sheet.nrows):
while '' in Arrivall[i]:
Arrivall[i].remove('')
```

```
for i in range(len(Project)):
Arrival.append(map(int,Arrivall[i]))
#print 'Arrival', Arrival

#
#
sheet = book.sheet_by_index(5)
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
   ,
 sheet.nrows, sheet.ncols)
PrecedenceN =[]
for i in xrange(sheet.nrows):
PrecedenceN.append(sheet.row_values(i))
for i in xrange(sheet.nrows):
while '' in PrecedenceN[i]:
PrecedenceN[i].remove('')
#print 'Precedence'

Precedence = []
z = 0
for n in Project:
n = z+n
x =  Precedence.append(PrecedenceN[z:n])
z = n
#print Precedence

#
#
sheet = book.sheet_by_index(13)
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
   ,
 sheet.nrows, sheet.ncols)
RS = []
for i in xrange(sheet.nrows):
```

```
RS.append(sheet.row_values(i))
for i in xrange(sheet.nrows):
while '' in RS[i]:
RS[i].remove('')
#print 'RemainingStatus', RS

#
#

# Auxilary
TotProj=len(Project)
IAux = range(1,TotProj+1)
JAux= dict(zip(IAux,(range(1,Project[i-1]+1) for i in IAux)))
#
#
Duration=[]
for i in IAux:
InflatedDuration=[]
for j in JAux[i]:
InflatedDuration.append(round(ActualDuration[i-1][j-1]*
   InflationRate
[i-1][j-1]))
Duration.append(InflatedDuration)


#
#
for i in IAux:
for j in JAux[i]:
if RS[i-1][j-1] > 0:
Duration[i-1][j-1]=(round(Duration[i-1][j-1]*RS[i-1][j
   -1]+0.4999))
else:
continue
```

```
#print 'Duration', Duration


#print 'DoubleCheckDuration',Duration
#
#
sheet = book.sheet_by_index(12)
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
   ,
sheet.nrows, sheet.ncols)
Batch = []
for i in xrange(sheet.nrows):
Batch.append(sheet.row_values(i))
for i in xrange(sheet.nrows):
while '' in Batch[i]:
Batch[i].remove('')
#print 'Batch', Batch

#
#
l= [[] for x in xrange(len(Project))]
def EF(n,j):
Templl= Arrival[n][j] + Duration[n][j]
for k in range(Project[n]):
if Precedence[n][j][k]==1:
temp = EF(n,k) - Duration[n][k] + round((Duration[n][k]/Batch[n
   ][k])
+0.49999) + Duration [n][j]
tempp= EF(n,k) + round((Duration[n][j]/Batch[n][j])+0.49999)
if temp > Templl:
Templl = temp
if tempp > Templl: Templl = tempp
else:
if tempp > Templl:
```

```
Templl = tempp
return Templl

n = 0
while n<len(Project):
for j in range(Project[n]):
l[n].append(int(EF(n,j)))
n+=1
#print 'Earliest Finish'
#print l

#
#
def LF(n,j):
TempU= AbsoluteDueDate[n]
for k in range(Project[n]):
if Precedence[n][k][j]==1:
temp = LF(n,k) - Duration[n][k]
if temp < TempU:  TempU = temp
return TempU
u= [[] for x in xrange(len(Project))]
n = 0
while n<len(Project):
for j in range(Project[n]):
u[n].append(int(LF(n,j)))
n+=1
#print 'Latest Finish'
#print u

#
#

sheet = book.sheet_by_index(8)
```

```
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
    ,
sheet.nrows, sheet.ncols)
Operatorr =[]
for i in xrange(sheet.nrows):
Operatorr.append(sheet.row_values(i))
OperatorReq = []
z = 0
for n in Project:
n = z+n
x =  OperatorReq.append(Operatorr[z:n])
z = n
#print 'OperatorReq'
#print OperatorReq

#
#
sheet = book.sheet_by_index(14)
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
    ,
sheet.nrows, sheet.ncols)
CC = []
CCC = []
for i in xrange(sheet.nrows):
CCC.append(sheet.row_values(i))
for i in xrange(sheet.nrows):
while '' in CCC[i]:
CCC[i].remove('')
for i in range(len(Project)):
CC.append(map(int,CCC[i]))
#print 'CompletionConstraint'
#print CC
#
#
```

```
sheet = book.sheet_by_index(9)
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
   ,
sheet.nrows, sheet.ncols)
OperatorPartialUsage =[]
for i in xrange(sheet.nrows):
OperatorPartialUsage.append(sheet.row_values(i))
Opu = []
z = 0
for n in Project:
n = z+n
x =  Opu.append(OperatorPartialUsage[z:n])
z = n
#print 'OperatorPartialUsage'
#print Opu

#
#
sheet = book.sheet_by_index(7)
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
   ,
 sheet.nrows, sheet.ncols)
MachinePartialUsage =[]
for i in xrange(sheet.nrows):
MachinePartialUsage.append(sheet.row_values(i))
Mpu = []
z = 0
for n in Project:
n = z+n
x =  Mpu.append(MachinePartialUsage[z:n])
z = n
#print 'MachinePartialUsage'
#print Mpu
```

```
#
#
Machinee = []
sheet = book.sheet_by_index(6)
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
   ,
sheet.nrows, sheet.ncols)
for i in xrange(sheet.nrows):
Machinee.append(sheet.row_values(i))
MachineReq = []
z = 0
for n in Project:
n = z+n
x =  MachineReq.append(Machinee[z:n])
z = n
#print 'MachineReq'
#print MachineReq

#
#
sheet = book.sheet_by_index(15)
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
   ,
sheet.nrows, sheet.ncols)
MC = []
MCC = []
for i in xrange(sheet.nrows):
MCC.append(sheet.row_values(i))
for i in xrange(sheet.nrows):
while '' in MCC[i]:
MCC[i].remove('')
for i in range(len(Project)):
MC.append(map(int,MCC[i]))
#print 'MachineConstraint'
```

```
#print MC
#
#
sheet = book.sheet_by_index(16)
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
   ,
 sheet.nrows, sheet.ncols)
OC = []
OCC=[]
for i in xrange(sheet.nrows):
OCC.append(sheet.row_values(i))
for i in xrange(sheet.nrows):
while '' in OCC[i]:
OCC[i].remove('')
for i in range(len(Project)):
OC.append(map(int,OCC[i]))
#print 'OperatorConstraint'
#print OC
#
#
sheet = book.sheet_by_index(17)
#print 'Worksheet Names: %s; Rows: %s; Columns: %s'% (sheet.name
   ,
sheet.nrows, sheet.ncols)
LFC = []
for i in xrange(sheet.nrows):
LFC.append(sheet.row_values(i))
for i in xrange(sheet.nrows):
while '' in LFC[i]:
LFC[i].remove('')
#print 'LastestFinishConstranit'
#print LFC
#
```

```
# Import PuLP modeler functions
from pulp import *

# Create the 'prob' variable to contain the problem data
prob = LpProblem("The Optimzation Problem",LpMinimize)

###-----Sets-----###
# number of projects
ProjNum=len(Project)
#set of project numbers
I = range(1,ProjNum+1)


#index auxilary
N=[[] for x in xrange(len(Project))]
for i in I:
N[i-1].append(i)

#number of jobs in Project i
P = dict(zip(I,((Project[i-1]) for i in I)))

#set of jobs in project i
J= dict(zip(I,(range(1,Project[i-1]+1) for i in I)))


#number of machine types
Machine=22
# set of machine types
M= range(1,Machine+1)

#number of operator types
Operator=14
```

```
#set of operators types
O= range(1,Operator+1)

for i in I:
for j in J[i]:
if Duration[i-1][j-1]==0:
RS[i-1][j-1]=float(0)


if Process=='Planning':
DurationPortion=[]
for i in I:
DP=[]
AbsoluteDueDate[i-1]=int(round((AbsoluteDueDate[i-1]/8.0)
   +0.4999))
DesiredDueDate[i-1]=int(round((DesiredDueDate[i-1]/8.0)+0.4999))
for j in J[i]:
if Duration[i-1][j-1]<>0:
DP.append(Duration[i-1][j-1]/8.0)
Arrival[i-1][j-1]= int(round((Arrival[i-1][j-1]/8.0)+0.4999))
for m in M:
Mpu[i-1][j-1][m-1]=Mpu[i-1][j-1][m-1]*((Duration[i-1][j-1]/8.0)/
(round((Duration[i-1][j-1]/8.0)+0.4999)))
for o in O:
Opu[i-1][j-1][o-1]=Opu[i-1][j-1][o-1]*((Duration[i-1][j-1]/8.0)/
(round((Duration[i-1][j-1]/8.0)+0.4999)))
Duration[i-1][j-1]=round((Duration[i-1][j-1]/8.0)+0.4999)
else:
Duration[i-1][j-1]=0.001
DP.append(Duration[i-1][j-1]/8.0)
Arrival[i-1][j-1]= int(round((Arrival[i-1][j-1]/8.0)+0.4999))
for m in M:
Mpu[i-1][j-1][m-1]=Mpu[i-1][j-1][m-1]*((Duration[i-1][j-1]/8.0)/
(round((Duration[i-1][j-1]/8.0)+0.4999)))
```

```
for o in O:
Opu[i-1][j-1][o-1]=Opu[i-1][j-1][o-1]*((Duration[i-1][j-1]/8.0)/
(round((Duration[i-1][j-1]/8.0)+0.4999)))
Duration[i-1][j-1]=round((Duration[i-1][j-1]/8.0)+0.4999)
DurationPortion.append(DP)
#print 'DurationPortion',DurationPortion
def LFP(n,j):
TempU= AbsoluteDueDate[n]
for k in range(P[n+1]):
if Precedence[n][k][j]==1:
temp = LFP(n,k) - DurationPortion[n][k]
if temp < TempU:  TempU = temp
return TempU
n = 0
u= [[] for x in xrange(len(P))]
while n<len(P):
for j in range(P[n+1]):
u[n].append(int(round(LFP(n,j)+0.4999)))
n+=1


def EFP(n,j):
Templl= Arrival[n][j] + DurationPortion[n][j]
for k in range(P[n+1]):
if Precedence[n][j][k]==1:
temp = EFP(n,k) - DurationPortion[n][k] + (DurationPortion[n][k
   ]/
Batch[n][k])+DurationPortion[n][j]
tempp= EFP(n,k) + DurationPortion[n][j]/Batch[n][j]
if temp > Templl:
Templl = temp
if tempp > Templl: Templl = tempp
else:
if tempp > Templl:
```

```
Templl = tempp
return Templl
n = 0
l= [[] for x in xrange(len(P))]
while n<len(P):
for j in range(P[n+1]):
l[n].append(int(round(EFP(n,j)+0.4999)))
n+=1

for i in I:
deletlist=[]
for j in range(Project[i-1]):
if RS[i-1][j]==0:
for k in range(Project[i-1]):
Precedence[i-1][k][j]=0
for z in range(Project[i-1]):
Precedence[i-1][j][z]=0
if Process=='Planning':
EFP(i-1,k)
l[i-1][k]=int(round(EFP(i-1,k)+0.4999))
LFP(i-1,k)
u[i-1][k]=int(round(LFP(i-1,k)+0.4999))
else:
EF(i-1,k)
l[i-1][k]=int(round(EF(i-1,k)+0.4999))
LF(i-1,k)
u[i-1][k]=int(round(LF(i-1,k)+0.4999))
deletlist.append(j+1)
else:
continue

for x in deletlist:
J[i].remove(x)
```

```
#print 'Latest Finish'
#print u
#print 'Earliest Finish'
#print l

for i in I:
if len(J[i])<=P[i]:
P[i]=len(J[i])
else:
continue
deletjob=[]
for i in I:
if not J[i]:
deletjob.append(i)
for x in deletjob:
I.remove(x)

if Process=='Planning':
for i in I:
for j in J[i]:
if LFC[i-1][j-1]>0:
LFC[i-1][j-1]=round(LFC[i-1][j-1]/8+0.4999)
if LFC[i-1][j-1]<=u[i-1][j-1]:
u[i-1][j-1]=int(LFC[i-1][j-1])

if Process<>'Planning':
for i in I:
for j in J[i]:
if LF(i-1,j-1)>0:
if LF(i-1,j-1)<=u[i-1][j-1]:
u[i-1][j-1]=int(LF(i-1,j-1))
ADD=[]
for i in I:
ADD.append(AbsoluteDueDate[i-1])
```

```
print '*DATA ARE IMPORTED...'
#earliest possible period in which project i could be completed
ee=[]
for i in I:
x=[]
for j in J[i]:
x.append(l[i-1][j-1])
eee=max(x)
ee.append(eee)
e=dict(zip(I,ee))


#priority of project i
w= dict(zip(I,Weight))

#time interval 1

T= dict(zip(I,(range(min(Arrival[i-1]),AbsoluteDueDate[i-1]+1)
   for i in I)))

#time interval 2
T2= range(0,max(ADD)+1)

#time interval 3
T3= {}
for i in I:
for j in J[i]: T3[i,j]= range(l[i-1][j-1],u[i-1][j-1]+1)

#time interval 4
T4={}
for i in I:
```

```
for j in J[i]: T4[i,j]= range(l[i-1][j-1],AbsoluteDueDate[i
    -1]+1)

#time interval 5
T5={}
for i in I:
for j in J[i]: T5[i,j]= range(l[i-1][j-1],u[i-1][j-1]+1)

#time interval 6
TeG={}
for i in I: TeG[i]= range(e[i],AbsoluteDueDate[i-1]+1)

#time interval 7
Tal={}
for i in I:
for j in J[i]: Tal[i,j]= range(Arrival[i-1][j-1],l[i-1][j-1])

#time interval 8
TuG={}
for i in I:
for j in J[i]: TuG[i,j]= range(u[i-1][j-1]+1,AbsoluteDueDate[i
    -1]+1)

#time interval 9
T1e= dict(zip(I,(range(min(Arrival[i-1]),e[i]) for i in I)))

R={}
for m in M:
for t in T2: R[m,t]=1
V={}
for o in O:
for t in T2: V[o,t]=1
```

```
#desired due date for project i
g= dict(zip(I,((DesiredDueDate[i-1]) for i in I)))


# a variavle which is 1 if job j in project i is completed in
   period t,
0 otherwise
x={}
for i in I: x[i] = LpVariable.dicts('x',(N[i-1],J[i],T[i]),0,1,
   LpInteger)

# a variable which is 1 in period t if all jobs of project i
   have been
comlpeted, 0 otherwise
h={}
for i in I: h[i] = LpVariable.dicts('h',(N[i-1],TeG[i]),0,1,
   LpInteger)

# a varibale which is 1 within a duration of job j in project i
zz={}
for i in I: zz[i] = LpVariable.dicts('zz',(N[i-1],J[i],T[i])
   ,0,1,LpInteger)

# a variable which is 1 if machine m is chosen for job j in
   project i,
0 otherwise
S={}
for i in I: S[i] = LpVariable.dicts('S',(N[i-1],J[i],M),0,1,
   LpInteger)

# a variable which  is 1 if operator n is chosen for job j in
   project i,
 0 otherwise
W={}
```

```
for i in I: W[i] = LpVariable.dicts('W',(N[i-1],J[i],O),0,1,
   LpInteger)

# a variable which is used to assisst constraint 5
QS={}
for i in I: QS[i] = LpVariable.dicts('QS',(N[i-1],J[i],M,T[i])
   ,0,1,LpInteger)

# a variable which is used to assisst constraint 6
QW={}
for i in I: QW[i] = LpVariable.dicts('QW',(N[i-1],J[i],O,T[i])
   ,0,1,
LpInteger)

# a variable which is 1 during the period that a setup for a
   step is
completed but the processing has not been
started yet - This is only used for machine utilization
zzs={}
for i in I: zzs[i] = LpVariable.dicts('zzs',(N[i-1],J[i],T[i])
   ,0,1
,LpInteger)

# a variable which is 1 if a machine is utilized during a period
    defined for zzs
QQS={}
for i in I: QQS[i] = LpVariable.dicts('QQS',(N[i-1],J[i],M,T[i])
   ,0,1,LpInteger)

Time_Start = time.clock()
print '*VARIABLES ARE GENERATED, THE MODEL IS BEING PROCESSED...
   ','     ##',Process,'##'
```

Appendix 2 ILP Model

```
#-----Objective Function-----#
# a project is late if it is completed after the desired due-
   date, g[i]
prob += (sum(w[i]*((t-g[i])*h[i][i][t]) for i in I for t in TeG[
   i]
if t>=g[i]+1)+ sum(0.01*w[i]*h[i][i][t]*t
 for i in I for t in TeG[i]))


#-----Subject To-----#
for i in I:
#job completion constraint 1
prob += sum(h[i][i][t] for t in TeG[i]) == 1
#prob += sum(h[i][i][t] for t in T1e[i]) == 0

for j in J[i]:
#step completion constraint
prob += sum(x[i][i][j][t] for t in T3[i,j] ) == 1
prob += sum(x[i][i][j][t] for t in TuG[i,j]) == 0
prob += sum(x[i][i][j][t] for t in Tal[i,j]) == 0
if CC[i-1][j-1]>0:
B = CC[i-1][j-1]
prob += x[i][i][j][B]==1


#step duration
for t in T[i]:
prob += sum(x[i][i][j][t1] for t1 in T[i] if t1>=t if t1< t +
   Duration[i-1]
[j-1] if t1 <= AbsoluteDueDate[i-1]) == zz[i][i][j][t]
#prob += sum(zz[i][i][j][t] for t in T[i] if t>= AbsoluteDueDate
   [i-1]+1)
 == 0
```

```
#prob += sum(zz[i][i][j][t] for t in Tal[i,j] if Duration[i-1][j
   -1]<t-
Duration[i-1][j-1]) == 0

#machine constraint auxilary
prob += sum(MachineReq[i-1][j-1][m-1] for m in M) >= sum(S[i][i
   ][j][m]
 for m in M)
prob += sum(MachineReq[i-1][j-1][m-1] for m in M) <= sum(S[i][i
   ][j][m]
for m in M)*Machine
prob += sum(S[i][i][j][m] for m in M) <= 1
for m in M:
prob += S[i][i][j][m] <= MachineReq[i-1][j-1][m-1]
for t in T[i]:
prob += zz[i][i][j][t]+S[i][i][j][m] - QS[i][i][j][m][t]*2 <= 1
prob += zz[i][i][j][t]+S[i][i][j][m] - QS[i][i][j][m][t]*2 >= 0
#machine constraint during a period, starting from setup
   completion
upto a begining of a corresponding processing
prob += zzs[i][i][j][t]+S[i][i][j][m] - QQS[i][i][j][m][t]*2 <=
   1
prob += zzs[i][i][j][t]+S[i][i][j][m] - QQS[i][i][j][m][t]*2 >=
   0
if MC[i-1][j-1]>0:
Y = MC[i-1][j-1]
prob += S[i][i][j][Y]==1

#operator constraint auxilary
prob += sum(OperatorReq[i-1][j-1][o-1] for o in O) >= sum(W[i][i
   ][j][o]
 for o in O)
prob += sum(OperatorReq[i-1][j-1][o-1] for o in O) <= sum(W[i][i
   ][j][o]
```

```
 for o in O)*Operator
prob += sum(W[i][i][j][o] for o in O) <= 1
for o in O:
prob += W[i][i][j][o] <= OperatorReq[i-1][j-1][o-1]
for t in T[i]:
prob += zz[i][i][j][t]+W[i][i][j][o] - QW[i][i][j][o][t]*2 <= 1
prob += zz[i][i][j][t]+W[i][i][j][o] - QW[i][i][j][o][t]*2 >= 0
if OC[i-1][j-1]>0:
B = OC[i-1][j-1]
prob += W[i][i][j][B]==1
#cycle must be assigned to a machine that is assigned to its
   corresponding
 setup
for j in J[i]:
if j%2<>0:
if j+1 in J[i]:
for m in M:
prob += S[i][i][j][m]==S[i][i][j+1][m]

#Machine constraint during the period in which the setup of a
   step is
completed but the cycle has not been
started yet
for j in J[i]:
if j%2<>0:
if j+1 in J[i]:
for t in T[i]:
if Process=='Planning':
if l[i-1][j-1]==l[i-1][j]:
prob += (sum(x[i][i][j][t1] for t1 in T[i] if t1>=l[i-1][j-1] if
    t1<=t)-sum
(x[i][i][j+1][t1] for t1 in T[i] if t1>=l[i-1][j] if t1<=t+
   Duration[i-1][j]))==zzs[i][i][j][t]
else:
```

```
prob += (sum(x[i][i][j][t1] for t1 in T[i] if t1>=l[i-1][j-1] if
    t1<t)-sum(
x[i][i][j+1][t1] for t1 in T[i] if t1>=l[i-1][j-1] if t1<t+
   Duration[i-1][j]))==zzs[i][i][j][t]
else:
prob += sum(x[i][i][j][t1] for t1 in T[i] if t1>=l[i-1][j-1] if
   t1<t)-sum
(x[i][i][j+1][t1] for t1 in T[i] if t1>l[i-1][j-1] if t1<t+
   Duration[i-1][j])==zzs[i][i][j][t]


#project completion constraint 2
for t2 in TeG[i]:
prob += sum(x[i][i][j][t] for j in J[i] for t in T3[i,j] if t<=
   t2) >= (h[i][i]
[t2])*P[i]


#Sequencing constraint
#for j1 in J[i]:
#    for j2 in J[i]:
#        prob += Precedence[i-1][j1-1][j2-1]*(sum(t*x[i][i][j2][
   t] for t in T5[i,j2])
+Duration[i-1][j1-1]) <= sum(t*x[i][i][j1][t] for t in T5[i,j1])
for j1 in J[i]:
for j2 in J[i]:
if Process=='Planning':
#prob += Precedence[i-1][j1-1][j2-1]*(sum(t*x[i][i][j2][t] for t
    in T5[i,j2])-D
uration[i-1][j2-1]+DurationPortion[i-1][j2-1])<=sum(t*x[i][i][j1
   ][t] for t in T5[i,j1])-DurationPortion[i-1][j1-1]
```

```
prob += Precedence[i-1][j1-1][j2-1]*(sum(t*x[i][i][j2][t] for t
   in T5[i,j2])-Duration[i-1][j2-1]+(DurationPortion[i-1][j2-1]/
   Batch[i-1][j2-1])) <= sum(t*x[i][i][j1][t] for t in T5[i,j1])
   -DurationPortion[i-1][j1-1]
prob += Precedence[i-1][j1-1][j2-1]*(sum(t*x[i][i][j2][t] for t
   in T5[i,j2])-Duration[i-1][j2-1]+DurationPortion[i-1][j2-1]+(
   DurationPortion[i-1][j1-1]/Batch[i-1][j1-1]))
 <= sum(t*x[i][i][j1][t] for t in T5[i,j1])
else:
prob += Precedence[i-1][j1-1][j2-1]*(sum(t*x[i][i][j2][t] for t
   in T5[i,j2])
-Duration[i-1][j2-1]+(Duration[i-1][j2-1]/Batch[i-1][j2-1])) <=
   sum(t*x[i][i][j1][t]
 for t in T5[i,j1])-Duration[i-1][j1-1]
prob += Precedence[i-1][j1-1][j2-1]*(sum(t*x[i][i][j2][t] for t
   in T5[i,j2])+
(Duration[i-1][j1-1]/Batch[i-1][j1-1]))
 <= sum(t*x[i][i][j1][t] for t in T5[i,j1])



#machine and operator constraint
for t in T2:
for m in M:
prob += (sum(QS[i][i][j][m][t]*Mpu[i-1][j-1][m-1] for i in I for
    j in J[i] if t in T[i])+
 sum(QQS[i][i][j][m][t]for i in I for j in J[i] if t in T[i]))<=
     R[m,t]

for o in O:
prob += sum(QW[i][i][j][o][t]*Opu[i-1][j-1][o-1] for i in I for
   j in J[i] if t in T[i])<= V[o,t]

#prob.writeLP("OptimizationProblem.lp")
```

```
#Gurobi Solver
print '*LP MODEL IS GENERATED----Solver: GUROBI'
solvers.GUROBI_CMD(path=None,keepFiles=0,mip=1,msg=1,options=[])
   .solve(prob)# options=
['TimeLimit=10']).solve(prob)

#GLPK Solver
#solvers.GLPK_CMD(path=None,keepFiles=0,mip=1,msg=1,options=[]).
   solve(prob)# options=
['TimeLimit=10']).solve(prob)

#COIN Solver
#print 'Solver: COIN'
#prob.solve()

print("Status:", LpStatus[prob.status])
print("Objective = ", value(prob.objective))
Time_Elapsed = (time.clock() - Time_Start)
print 'Computation Time', Time_Elapsed

if LpStatus[prob.status]=='Not Solved':
#print 'The model could not provide a feasible solution'
sys.exit("The model could not provide a feasible solution")


print ""
JobCompletion = ""
print "#JOB COMPLETION TIME#"
for i in I:
JobCompletion = JobCompletion + "Job "+repr(i)+" Completion_Time
   :"
for t in TeG[i]:
if h[i][i][t].value()==1:
JobCompletion = JobCompletion + str(t)
```

```
else:
continue
print JobCompletion
JobCompletion = ""

print ""


CompletionTime = ""
print "#STEP COMPLETION TIME#"
for i in I:
print "****Job "+repr(i)+"****"
for j in J[i]:
CompletionTime= "   Step "+repr(j)+":  "
for t in T[i]:
if x[i][i][j][t].value()==1:
CompletionTime = CompletionTime +str(t)
else:
continue
print CompletionTime
CompletionTime = ""

print ""


ProcessingPeriods = ""
print "#JOB PROCESSING PERIODS#"
for i in I:
print "****Job "+repr(i)+"****"
for j in J[i]:
ProcessingPeriods = ProcessingPeriods+"   Step"+repr(j)+": "+"(
   "
for t in T[i]:
if zz[i][i][j][t].value()==1:
```

```
ProcessingPeriods = ProcessingPeriods + str(t)+" "
else:
continue
ProcessingPeriods=ProcessingPeriods + ")"
print ProcessingPeriods
ProcessingPeriods = ""

print ""


MachineUsagePeriod = ""
print "#Machine USAGE PERIOD#"
for m in M:
print "****Machine "+repr(m)+"****"
print "   Job - step - period "
for t in T2:
for i in I:
if t in T[i]:
for j in J[i]:
if QS[i][i][j][m][t].value()==1 or QQS[i][i][j][m][t].value()
   ==1:
print "    "+str(i)+"      "+str(j)+"      "+str(t)
MachineUsagePeriod ="1"
else:
continue
if MachineUsagePeriod=="":
print "***Machine "+repr(m)+" was not used AT ALL***"
else:
MachineUsagePeriod = ""

print ""


OperatorUsagePeriod = ""
```

```
print "#OPERATOR USAGE PERIOD#"
for o in O:
print "****Operator "+repr(o)+"****"
print "   Job - step - period "
for t in T2:
for i in I:
if t in T[i]:
for j in J[i]:
if QW[i][i][j][o][t].value()==1:
print "    "+str(i)+"      "+str(j)+"      "+str(t)
OperatorUsagePeriod="1"
else:
continue
if OperatorUsagePeriod=="":
print "***Operator "+repr(o)+" was not used AT ALL***"
else:
OperatorUsagePeriod = ""



##GanttChart##
Identitylist=[]
numpylist=[]
for i in I:
for j in J[i]:
for t in T[i]:
if zz[i][i][j][t].value()==1:
steplist=[]
steplist.append(i+0.05*j)
steplist.append(j)
steplist.append(t-1)
steplist.append(t)
Identitylist.append(str(i)+"-"+str(j))
numpylist.append(steplist)
```

```
else:
continue
#print 'numpylist',numpylist
ganttchart=np.array(numpylist)
np.savetxt("ganttchart.csv", ganttchart, delimiter=",")
np.savetxt("numpylist.csv", numpylist, delimiter=",")
file = open("Identitylist.txt", "w")
for item in Identitylist:
file.write("%s\n" % item)
file.close()


plt.ylim(0,max(I)+1)

color= ['r', 'b', 'g', 'k', 'm', 'c', 'y']
color_mapper = np.vectorize(lambda x: {1: 'r', 2: 'b', 3: 'g',
   4:'m', 5:'c', 6:'y', 7:
'r', 8:'b', 9:'g', 10:'k', 11:'m', 12:'c', 13:'y',14:'k'}.get(x)
   )


plt.hlines(ganttchart[:,0],ganttchart[:,2],ganttchart[:,3],
   colors=color_mapper
(ganttchart[:,1]),linewidth=5)
plt.title('Jobs vs. Time')
plt.ylabel('Jobs')
if Process=='Planning':
plt.xlabel('Time(days)')
else:
plt.xlabel('Time(hours)')
```

```
#plt.locator_params('both',1)
plt.yticks(range(0,max(I)+1))
plt.minorticks_on()
plt.tick_params(axis = 'both', which = 'major', labelsize = 10)
#plt.set_major_formatter(majorFormatter)


x1=0
x2=1
x3=2
x4=3
n=0
while n<len(numpylist):
x5=float(numpylist[n][x3]+numpylist[n][x4])/2
plt.text(x5,numpylist[n][x1],Identitylist[n])
n+=1

plt.show()

###

Identitymachine=[]
numpymachine=[]
for m in M:
for i in I:
for j in J[i]:
for t in T[i]:
if QS[i][i][j][m][t].value()==1:
steplist=[]
steplist.append(i)
steplist.append(j)
steplist.append(m+0.05*i)
steplist.append(t-1)
steplist.append(t)
```

```
Identitymachine.append(str(i)+"-"+str(j))
numpymachine.append(steplist)
else:
continue

#print 'numpymachine',numpymachine
ganttchartm=np.array(numpymachine)

np.savetxt("ganttchartm.csv", ganttchartm, delimiter=",")
np.savetxt("numpymachine.csv", numpymachine, delimiter=",")
file = open("Identitymachine.txt", "w")
for item in Identitymachine:
file.write("%s\n" % item)
file.close()


plt.yticks(range(len(M)+1))
plt.ylim(0,len(M)+2)
plt.hlines(ganttchartm[:,2],ganttchartm[:,3],ganttchartm[:,4],
   linewidth=4,colors
=color_mapper(ganttchartm[:,1]))
plt.title('Machine Utilization')
plt.ylabel('Machine')
if Process=='Planning':
plt.xlabel('Time(days)')
else:
plt.xlabel('Time(hours)')


x1=0
x2=1
x3=2
x4=3
x5=4
```

```
n=0
while n<len(numpymachine):
x6=float(((numpymachine[n][x4]+numpymachine[n][x5])/2)+0.05*
   numpymachine[n][x2])
plt.text(x6,numpymachine[n][x3],Identitymachine[n])
n+=1

plt.show()


###

Identityoperator=[]
numpyoperator=[]
for o in O:
for i in I:
for j in J[i]:
for t in T[i]:
if QW[i][i][j][o][t].value()==1:
steplist=[]
steplist.append(i)
steplist.append(j)
steplist.append(o+0.05*i)
steplist.append(t-1)
steplist.append(t)
Identityoperator.append(str(i)+"-"+str(j))
numpyoperator.append(steplist)
else:
continue
#print 'numpyoperator',numpyoperator
ganttcharto=np.array(numpyoperator)

np.savetxt("ganttcharto.csv", ganttcharto, delimiter=",")
np.savetxt("numpyoperator.csv", numpyoperator, delimiter=",")
```

```
file = open("Identityoperator.txt", "w")
for item in Identityoperator:
file.write("%s\n" % item)
file.close()

#color_mapper = np.vectorize(lambda x: {1: 'red', 2: 'blue', 3:
   'green'}.get(x))
plt.yticks(range(len(O)+1))
plt.ylim(0,len(O)+2)
plt.hlines(ganttcharto[:,2],ganttcharto[:,3],ganttcharto[:,4],
   linewidth=4,
colors=color_mapper(ganttcharto[:,1]))
plt.title('Operator Utilization')
plt.ylabel('Operator')
if Process=='Planning':
plt.xlabel('Time(days)')
else:
plt.xlabel('Time(hours)')

x1=0
x2=1
x3=2
x4=3
x5=4
n=0
while n<len(numpyoperator):
x6=float(((numpyoperator[n][x4]+numpyoperator[n][x5])/2)+0.05*
   numpyoperator[n][x2])
plt.text(x6,numpyoperator[n][x3],Identityoperator[n])
n+=1

plt.show()

##if Process=='Planning':
```

```
##    BusyPeriods=[]
##    SchedulingPeriod=[]
##    for t in T2:
##        SchedulingPeriod1=[]
##        for i in I:
##            if t in T[i]:
##                for j in J[i]:
##                    if zz[i][i][j][t].value()==1:
##                        if t not in BusyPeriods:
##                            BusyPeriods.append(t)
##                        Schedule=[]
##                        Schedule.append(str(i)+"-"+str(j))
##                        SchedulingPeriod1.append(Schedule)
##        SchedulingPeriod.append(SchedulingPeriod1)
prevName = 'datanew.xls'
if Process=='Planning':
newName = 'datanewPlanning.xls'
else:
newName = 'datanewScheduling.xls'

shutil.copyfile(prevName,newName)

print 'Computation Time', Time_Elapsed
```

Appendix 3 SPT Model

```
print '*VARIABLES ARE GENERATED, THE MODEL IS BEING PROCESSED...
   ','
    ##',Process,'##'



#JobsToComplete
JTC=[]
for i in I:
#DurationListJob
DLJ=[]
for j in J[i]:
#DurationListStep
DLS=[]
DLS.append(i)
DLS.append(j)
DLS.append(Duration[i-1][j-1])
JTC.append(DLS)


def LESPP(i,j):
for j2 in J[i]:
if Precedence[i-1][j-1][j2-1]==1:
return 0
else: continue
return 1

LES=[]
for i in I:
for j in J[i]:
if LESPP(i,j)==1:
LES.append([i,j,Duration[i-1][j-1]])
LES=sorted(LES,key=itemgetter(2))
```

```
def LESP(i,j):
for j2 in J[i]:
if X[i-1][j-1][j2-1]>t:
return 0
else:
if Precedence[i-1][j-1][j2-1]==1:
return 0
else: continue
return 1


#List of eligible steps
def LESD():
del LES[:]
for i in I:
for j in J[i]:
if [i,j,Duration[i-1][j-1]] in JTC:
if LESP(i,j)==1:
LES.append([i,j,Duration[i-1][j-1]])
else:
continue
else:
continue




Identitylist=[]
numpylist=[]
Identitymachine=[]
numpymachine=[]
Identityoperator=[]
```

```
numpyoperator=[]

#Machine Avilability
def MA(x1,x2,t):
D=range(t,t+int(Duration[x1-1][x2-1]))
for m in MRL[x1-1][x2-1]:
Mcheck=[]
for t in D:
if (R[m,t]-Mpu[x1-1][x2-1][m-1])>=0:
Mcheck.append(1)
else:
Mcheck.append(0)
if sum(Mcheck)==len(D):
return m
else:
continue
return 0

#Operator Vavailabilty
def OA(x1,x2,t):
D=range(t,t+int(Duration[x1-1][x2-1]))
for o in ORL[x1-1][x2-1]:
Ocheck=[]
for t in D:
if (V[o,t]-Opu[x1-1][x2-1][o-1])>=0:
Ocheck.append(1)
else:
Ocheck.append(0)
if sum(Ocheck)==len(D):
return o
else:
continue
return 0
```

```
#Machine Constraint Cycle
def MCC(x1,x2,m):
if x2%2<>0:
MRL[x1-1][x2]=[]
MRL[x1-1][x2].append(m)


def AC(x1,x2,t):
if t>=Arrival[x1-1][x2-1]:
D=range(t,t+int(Duration[x1-1][x2-1]))
if MA(x1,x2,t)>=1 and OA(x1,x2,t)>=1:
m=MA(x1,x2,t)
#print "MA(X1,X2,t)",x1,x2,m
o=OA(x1,x2,t)
#print "OA(X1,X2,t)",x1,x2,o
MCC(x1,x2,m)
for t in D:
R[m,t]=R[m,t]-Mpu[x1-1][x2-1][m-1]
V[o,t]=V[o,t]-Opu[x1-1][x2-1][o-1]
for j in J[x1]:
Precedence[x1-1][j-1][x2-1]=float(0)
X[x1-1][j-1][x2-1]=int(max(D))
steplist=[]
steplist.append(x1+0.01*x2)
steplist.append(x2)
steplist.append(min(D))
steplist.append(max(D)+1)
Identitylist.append(str(x1)+"-"+str(x2))
numpylist.append(steplist)
steplist=[]
steplist.append(x1+0.01*x2)
steplist.append(x2)
steplist.append(m+0.01*x1)
steplist.append(min(D))
```

```
steplist.append(max(D)+1)
Identitymachine.append(str(x1)+"-"+str(x2))
numpymachine.append(steplist)
steplist=[]
steplist.append(x1+0.01*x2)
steplist.append(x2)
steplist.append(o+0.01*x1)
steplist.append(min(D))
steplist.append(max(D)+1)
Identityoperator.append(str(x1)+"-"+str(x2))
numpyoperator.append(steplist)

return 1




for t in T2:
if JTC:
LES=sorted(LES,key=itemgetter(2))
for x in LES:
x1=x[0]
x2=x[1]
if AC(x1,x2,t)==1:
JTC.remove(x)
LESD()




ganttchart=np.array(numpylist)
np.savetxt("ganttchart.csv", ganttchart, delimiter=",")
np.savetxt("numpylist.csv", numpylist, delimiter=",")
```

```
file = open("Identitylist.txt", "w")
for item in Identitylist:
file.write("%s\n" % item)
file.close()


plt.ylim(0,max(I)+1)

color= ['r', 'b', 'g', 'k', 'm', 'c', 'y']
color_mapper = np.vectorize(lambda x: {1: 'r', 2: 'b', 3: 'g',
   4:'m',
5:'c', 6:'y', 7:'r', 8:'b', 9:'g', 10:'k', 11:'m', 12:'c', 13:'y
   ',14:'k'}.get(x))


plt.hlines(ganttchart[:,0],ganttchart[:,2],ganttchart[:,3],
colors=color_mapper(ganttchart[:,1]),linewidth=5)
plt.title('Jobs vs. Time')
plt.ylabel('Jobs')
if Process=='Planning':
plt.xlabel('Time(days)')
else:
plt.xlabel('Time(hours)')



#plt.locator_params('both',1)
plt.yticks(range(0,max(I)+1))
plt.minorticks_on()
plt.tick_params(axis = 'both', which = 'major', labelsize = 10)
#plt.set_major_formatter(majorFormatter)
```

```
x1=0
x2=1
x3=2
x4=3
n=0
while n<len(numpylist):
x5=float(numpylist[n][x3]+numpylist[n][x4])/2
plt.text(x5,numpylist[n][x1],Identitylist[n])
n+=1

plt.show()




#print 'numpymachine',numpymachine
ganttchartm=np.array(numpymachine)

np.savetxt("ganttchartm.csv", ganttchartm, delimiter=",")
np.savetxt("numpymachine.csv", numpymachine, delimiter=",")
file = open("Identitymachine.txt", "w")
for item in Identitymachine:
file.write("%s\n" % item)
file.close()


plt.yticks(range(len(M)+1))
plt.ylim(0,len(M)+2)
plt.hlines(ganttchartm[:,2],ganttchartm[:,3],ganttchartm[:,4],
linewidth=4,colors=color_mapper(ganttchartm[:,1]))
plt.title('Machine Utilization')
plt.ylabel('Machine')
if Process=='Planning':
```

```
plt.xlabel('Time(days)')
else:
plt.xlabel('Time(hours)')


x1=0
x2=1
x3=2
x4=3
x5=4
n=0
while n<len(numpymachine):
x6=float(((numpymachine[n][x4]+numpymachine[n][x5])/2)+0.05*
numpymachine[n][x2])
plt.text(x6,numpymachine[n][x3],Identitymachine[n])
n+=1

plt.show()



#print 'numpyoperator',numpyoperator
ganttcharto=np.array(numpyoperator)

np.savetxt("ganttcharto.csv", ganttcharto, delimiter=",")
np.savetxt("numpyoperator.csv", numpyoperator, delimiter=",")
file = open("Identityoperator.txt", "w")
for item in Identityoperator:
file.write("%s\n" % item)
file.close()

#color_mapper = np.vectorize(lambda x: {1: 'red', 2: 'blue', 3:
green'}.get(x))
```

```
plt.yticks(range(len(O)+1))
plt.ylim(0,len(O)+2)
plt.hlines(ganttcharto[:,2],ganttcharto[:,3],ganttcharto[:,4],
linewidth=4,colors=color_mapper(ganttcharto[:,1]))
plt.title('Operator Utilization')
plt.ylabel('Operator')
if Process=='Planning':
plt.xlabel('Time(days)')
else:
plt.xlabel('Time(hours)')

x1=0
x2=1
x3=2
x4=3
x5=4
n=0
while n<len(numpyoperator):
x6=float(((numpyoperator[n][x4]+numpyoperator[n][x5])/2)+0.05*
numpyoperator[n][x2])
plt.text(x6,numpyoperator[n][x3],Identityoperator[n])
n+=1


plt.show()
```

Appendix 4 EDD Model

```
print '*DATA ARE IMPORTED...'
#earliest possible period in which project i could be completed
ee=[]
for i in I:
x=[]
for j in J[i]:
x.append(l[i-1][j-1])
eee=max(x)
ee.append(eee)
e=dict(zip(I,ee))


#priority of project i
w= dict(zip(I,Weight))

#time interval 1

T= dict(zip(I,(range(min(Arrival[i-1]),AbsoluteDueDate[i-1]+1)
 for i in I)))

#time interval 2
T2= range(min(min(A)),max(ADD)+1)

#time interval 3
T3= {}
for i in I:
for j in J[i]: T3[i,j]= range(l[i-1][j-1],u[i-1][j-1]+1)

#time interval 4
T4={}
for i in I:
```

```
for j in J[i]: T4[i,j]= range(l[i-1][j-1],AbsoluteDueDate[i
    -1]+1)

#time interval 5
T5={}
for i in I:
for j in J[i]: T5[i,j]= range(l[i-1][j-1],u[i-1][j-1]+1)

#time interval 6
TeG={}
for i in I: TeG[i]= range(e[i],AbsoluteDueDate[i-1]+1)

#time interval 7
Tal={}
for i in I:
for j in J[i]: Tal[i,j]= range(Arrival[i-1][j-1],l[i-1][j-1])

#time interval 8
TuG={}
for i in I:
for j in J[i]: TuG[i,j]= range(u[i-1][j-1]+1,AbsoluteDueDate[i
    -1]+1)

#time interval 9
T1e= dict(zip(I,(range(min(Arrival[i-1]),e[i]) for i in I)))

R={}
for m in M:
for t in T2: R[m,t]=1
V={}
for o in O:
for t in T2: V[o,t]=1
```

```
#desired due date for project i
g= dict(zip(I,((DesiredDueDate[i-1]) for i in I)))

Time_Start = time.clock()
print '*VARIABLES ARE GENERATED, THE MODEL IS BEING PROCESSED...
   ','
     ##',Process,'##'



#JobsToComplete
JTC=[]
for i in I:
#DurationListJob
DLJ=[]
for j in J[i]:
#DurationListStep
DLS=[]
DLS.append(i)
DLS.append(j)
DLS.append(Duration[i-1][j-1])
DLS.append(DesiredDueDate[i-1])
JTC.append(DLS)


def LESPP(i,j):
for j2 in J[i]:
if Precedence[i-1][j-1][j2-1]==1:
return 0
else: continue
return 1

LES=[]
for i in I:
```

```
for j in J[i]:
if LESPP(i,j)==1:
LES.append([i,j,Duration[i-1][j-1],DesiredDueDate[i-1]])
LES=sorted(LES,key=itemgetter(3))


def LESP(i,j):
for j2 in J[i]:
if X[i-1][j-1][j2-1]>t:
return 0
else:
if Precedence[i-1][j-1][j2-1]==1:
return 0
else: continue
return 1


#List of eligible steps
def LESD():
del LES[:]
for i in I:
for j in J[i]:
if [i,j,Duration[i-1][j-1],DesiredDueDate[i-1]] in JTC:
if LESP(i,j)==1:
LES.append([i,j,Duration[i-1][j-1],DesiredDueDate[i-1]])
else:
continue
else:
continue



Identitylist=[]
numpylist=[]
```

```
Identitymachine=[]
numpymachine=[]
Identityoperator=[]
numpyoperator=[]

#Machine Avilability
def MA(x1,x2,t):
D=range(t,t+int(Duration[x1-1][x2-1]))
for m in MRL[x1-1][x2-1]:
Mcheck=[]
for t in D:
if (R[m,t]-Mpu[x1-1][x2-1][m-1])>=0:
Mcheck.append(1)
else:
Mcheck.append(0)
if sum(Mcheck)==len(D):
return m
else:
continue
return 0

#Operator Vavailabilty
def OA(x1,x2,t):
D=range(t,t+int(Duration[x1-1][x2-1]))
for o in ORL[x1-1][x2-1]:
Ocheck=[]
for t in D:
if (V[o,t]-Opu[x1-1][x2-1][o-1])>=0:
Ocheck.append(1)
else:
Ocheck.append(0)
if sum(Ocheck)==len(D):
return o
else:
```

```
continue
return 0

#Machine Constraint Cycle
def MCC(x1,x2,m):
if x2%2<>0:
MRL[x1-1][x2]=[]
MRL[x1-1][x2].append(m)


def AC(x1,x2,t):
if t>=Arrival[x1-1][x2-1]:
D=range(t,t+int(Duration[x1-1][x2-1]))
if MA(x1,x2,t)>=1 and OA(x1,x2,t)>=1:
m=MA(x1,x2,t)
#print "MA(X1,X2,t)",x1,x2,m
o=OA(x1,x2,t)
#print "OA(X1,X2,t)",x1,x2,o
MCC(x1,x2,m)
for t in D:
R[m,t]=R[m,t]-Mpu[x1-1][x2-1][m-1]
V[o,t]=V[o,t]-Opu[x1-1][x2-1][o-1]
for j in J[x1]:
Precedence[x1-1][j-1][x2-1]=float(0)
X[x1-1][j-1][x2-1]=int(max(D))
steplist=[]
steplist.append(x1+0.01*x2)
steplist.append(x2)
steplist.append(min(D))
steplist.append(max(D)+1)
Identitylist.append(str(x1)+"-"+str(x2))
numpylist.append(steplist)
steplist=[]
steplist.append(x1+0.01*x2)
```

```
steplist.append(x2)
steplist.append(m+0.01*x1)
steplist.append(min(D))
steplist.append(max(D)+1)
Identitymachine.append(str(x1)+"-"+str(x2))
numpymachine.append(steplist)
steplist=[]
steplist.append(x1+0.01*x2)
steplist.append(x2)
steplist.append(o+0.01*x1)
steplist.append(min(D))
steplist.append(max(D)+1)
Identityoperator.append(str(x1)+"-"+str(x2))
numpyoperator.append(steplist)

return 1




for t in T2:
if JTC:
LES=sorted(LES,key=itemgetter(3))
for x in LES:
x1=x[0]
x2=x[1]
if AC(x1,x2,t)==1:
JTC.remove(x)
LESD()
```

```
ganttchart=np.array(numpylist)
np.savetxt("ganttchart.csv", ganttchart, delimiter=",")
np.savetxt("numpylist.csv", numpylist, delimiter=",")
file = open("Identitylist.txt", "w")
for item in Identitylist:
file.write("%s\n" % item)
file.close()


plt.ylim(0,max(I)+1)

color= ['r', 'b', 'g', 'k', 'm', 'c', 'y']
color_mapper = np.vectorize(lambda x: {1: 'r', 2: 'b', 3: 'g', 4
:'m', 5:'c', 6:'y', 7:'r', 8:'b', 9:'g', 10:'k', 11:'m', 12:'c',
    13:'y',14:'k'}.get(x))


plt.hlines(ganttchart[:,0],ganttchart[:,2],ganttchart[:,3],
colors=color_mapper(ganttchart[:,1]),linewidth=5)
plt.title('Jobs vs. Time')
plt.ylabel('Jobs')
if Process=='Planning':
plt.xlabel('Time(days)')
else:
plt.xlabel('Time(hours)')




#plt.locator_params('both',1)
plt.yticks(range(0,max(I)+1))
plt.minorticks_on()
plt.tick_params(axis = 'both', which = 'major', labelsize = 10)
#plt.set_major_formatter(majorFormatter)
```

```
x1=0
x2=1
x3=2
x4=3
n=0
while n<len(numpylist):
x5=float(numpylist[n][x3]+numpylist[n][x4])/2
plt.text(x5,numpylist[n][x1],Identitylist[n])
n+=1

plt.show()




#print 'numpymachine',numpymachine
ganttchartm=np.array(numpymachine)

np.savetxt("ganttchartm.csv", ganttchartm, delimiter=",")
np.savetxt("numpymachine.csv", numpymachine, delimiter=",")
file = open("Identitymachine.txt", "w")
for item in Identitymachine:
file.write("%s\n" % item)
file.close()


plt.yticks(range(len(M)+1))
plt.ylim(0,len(M)+2)
plt.hlines(ganttchartm[:,2],ganttchartm[:,3],ganttchartm[:,4],
linewidth=4,colors=color_mapper(ganttchartm[:,1]))
plt.title('Machine Utilization')
```

```
plt.ylabel('Machine')
if Process=='Planning':
plt.xlabel('Time(days)')
else:
plt.xlabel('Time(hours)')


x1=0
x2=1
x3=2
x4=3
x5=4
n=0
while n<len(numpymachine):
x6=float(((numpymachine[n][x4]+numpymachine[n][x5])/2)+0.05*
numpymachine[n][x2])
plt.text(x6,numpymachine[n][x3],Identitymachine[n])
n+=1

plt.show()



#print 'numpyoperator',numpyoperator
ganttcharto=np.array(numpyoperator)

np.savetxt("ganttcharto.csv", ganttcharto, delimiter=",")
np.savetxt("numpyoperator.csv", numpyoperator, delimiter=",")
file = open("Identityoperator.txt", "w")
for item in Identityoperator:
file.write("%s\n" % item)
file.close()
```

```
#color_mapper = np.vectorize(lambda x: {1: 'red', 2: 'blue',
3: 'green'}.get(x))
plt.yticks(range(len(O)+1))
plt.ylim(0,len(O)+2)
plt.hlines(ganttcharto[:,2],ganttcharto[:,3],ganttcharto[:,4]
,linewidth=4,colors=color_mapper(ganttcharto[:,1]))
plt.title('Operator Utilization')
plt.ylabel('Operator')
if Process=='Planning':
plt.xlabel('Time(days)')
else:
plt.xlabel('Time(hours)')

x1=0
x2=1
x3=2
x4=3
x5=4
n=0
while n<len(numpyoperator):
x6=float(((numpyoperator[n][x4]+numpyoperator[n][x5])/2)+
0.05*numpyoperator[n][x2])
plt.text(x6,numpyoperator[n][x3],Identityoperator[n])
n+=1


plt.show()
```

Appendix 5 SPT-EDD Model

```
print '*DATA ARE IMPORTED...'
#earliest possible period in which project i could be
completed
ee=[]
for i in I:
x=[]
for j in J[i]:
x.append(l[i-1][j-1])
eee=max(x)
ee.append(eee)
e=dict(zip(I,ee))


#priority of project i
w= dict(zip(I,Weight))

#time interval 1

T= dict(zip(I,(range(min(Arrival[i-1]),AbsoluteDueDate
[i-1]+1) for i in I)))

#time interval 2
T2= range(min(min(A)),max(ADD)+1)

#time interval 3
T3= {}
for i in I:
for j in J[i]: T3[i,j]= range(l[i-1][j-1],u[i-1][j-1]+1)

#time interval 4
T4={}
for i in I:
```

```
for j in J[i]: T4[i,j]= range(l[i-1][j-1],AbsoluteDueDate[i
    -1]+1)

#time interval 5
T5={}
for i in I:
for j in J[i]: T5[i,j]= range(l[i-1][j-1],u[i-1][j-1]+1)

#time interval 6
TeG={}
for i in I: TeG[i]= range(e[i],AbsoluteDueDate[i-1]+1)

#time interval 7
Tal={}
for i in I:
for j in J[i]: Tal[i,j]= range(Arrival[i-1][j-1],l[i-1][j-1])

#time interval 8
TuG={}
for i in I:
for j in J[i]: TuG[i,j]= range(u[i-1][j-1]+1,AbsoluteDueDate[i
    -1]+1)

#time interval 9
T1e= dict(zip(I,(range(min(Arrival[i-1]),e[i]) for i in I)))

R={}
for m in M:
for t in T2: R[m,t]=1
V={}
for o in O:
for t in T2: V[o,t]=1
```

```
#desired due date for project i
g= dict(zip(I,((DesiredDueDate[i-1]) for i in I)))

Time_Start = time.clock()
print '*VARIABLES ARE GENERATED, THE MODEL IS BEING PROCESSED...
   ','
     ##',Process,'##'



#JobsToComplete
JTC=[]
for i in I:
#DurationListJob
DLJ=[]
for j in J[i]:
#DurationListStep
DLS=[]
DLS.append(i)
DLS.append(j)
DLS.append(Duration[i-1][j-1])
DLS.append(DesiredDueDate[i-1])
JTC.append(DLS)


def LESPP(i,j):
for j2 in J[i]:
if Precedence[i-1][j-1][j2-1]==1:
return 0
else: continue
return 1

LES=[]
for i in I:
```

```
for j in J[i]:
if LESPP(i,j)==1:
LES.append([i,j,Duration[i-1][j-1],DesiredDueDate[i-1]])
LES=sorted(LES,key=itemgetter(2,3))


def LESP(i,j):
for j2 in J[i]:
if X[i-1][j-1][j2-1]>t:
return 0
else:
if Precedence[i-1][j-1][j2-1]==1:
return 0
else: continue
return 1


#List of eligible steps
def LESD():
del LES[:]
for i in I:
for j in J[i]:
if [i,j,Duration[i-1][j-1],DesiredDueDate[i-1]] in JTC:
if LESP(i,j)==1:
LES.append([i,j,Duration[i-1][j-1],DesiredDueDate[i-1]])
else:
continue
else:
continue



Identitylist=[]
numpylist=[]
```

```
Identitymachine=[]
numpymachine=[]
Identityoperator=[]
numpyoperator=[]

#Machine Avilability
def MA(x1,x2,t):
D=range(t,t+int(Duration[x1-1][x2-1]))
for m in MRL[x1-1][x2-1]:
Mcheck=[]
for t in D:
if (R[m,t]-Mpu[x1-1][x2-1][m-1])>=0:
Mcheck.append(1)
else:
Mcheck.append(0)
if sum(Mcheck)==len(D):
return m
else:
continue
return 0

#Operator Vavailabilty
def OA(x1,x2,t):
D=range(t,t+int(Duration[x1-1][x2-1]))
for o in ORL[x1-1][x2-1]:
Ocheck=[]
for t in D:
if (V[o,t]-Opu[x1-1][x2-1][o-1])>=0:
Ocheck.append(1)
else:
Ocheck.append(0)
if sum(Ocheck)==len(D):
return o
else:
```

```
continue
return 0

#Machine Constraint Cycle
def MCC(x1,x2,m):
if x2%2<>0:
MRL[x1-1][x2]=[]
MRL[x1-1][x2].append(m)


def AC(x1,x2,t):
if t>=Arrival[x1-1][x2-1]:
D=range(t,t+int(Duration[x1-1][x2-1]))
if MA(x1,x2,t)>=1 and OA(x1,x2,t)>=1:
m=MA(x1,x2,t)
#print "MA(X1,X2,t)",x1,x2,m
o=OA(x1,x2,t)
#print "OA(X1,X2,t)",x1,x2,o
MCC(x1,x2,m)
for t in D:
R[m,t]=R[m,t]-Mpu[x1-1][x2-1][m-1]
V[o,t]=V[o,t]-Opu[x1-1][x2-1][o-1]
for j in J[x1]:
Precedence[x1-1][j-1][x2-1]=float(0)
X[x1-1][j-1][x2-1]=int(max(D))
steplist=[]
steplist.append(x1+0.01*x2)
steplist.append(x2)
steplist.append(min(D))
steplist.append(max(D)+1)
Identitylist.append(str(x1)+"-"+str(x2))
numpylist.append(steplist)
steplist=[]
steplist.append(x1+0.01*x2)
```

```
steplist.append(x2)
steplist.append(m+0.01*x1)
steplist.append(min(D))
steplist.append(max(D)+1)
Identitymachine.append(str(x1)+"-"+str(x2))
numpymachine.append(steplist)
steplist=[]
steplist.append(x1+0.01*x2)
steplist.append(x2)
steplist.append(o+0.01*x1)
steplist.append(min(D))
steplist.append(max(D)+1)
Identityoperator.append(str(x1)+"-"+str(x2))
numpyoperator.append(steplist)

return 1



for t in T2:
if JTC:
LES=sorted(LES,key=itemgetter(2,3))
for x in LES:
x1=x[0]
x2=x[1]
if AC(x1,x2,t)==1:
JTC.remove(x)
LESD()
```

```
ganttchart=np.array(numpylist)
np.savetxt("ganttchart.csv", ganttchart, delimiter=",")
np.savetxt("numpylist.csv", numpylist, delimiter=",")
file = open("Identitylist.txt", "w")
for item in Identitylist:
file.write("%s\n" % item)
file.close()


plt.ylim(0,max(I)+1)

color= ['r', 'b', 'g', 'k', 'm', 'c', 'y']
color_mapper = np.vectorize(lambda x: {1: 'r', 2: 'b', 3: 'g',
   4:'m'
, 5:'c', 6:'y', 7:'r', 8:'b', 9:'g', 10:'k', 11:'m', 12:'c', 13:
   'y',14:'k'}.get(x))


plt.hlines(ganttchart[:,0],ganttchart[:,2],ganttchart[:,3],
colors=color_mapper(ganttchart[:,1]),linewidth=5)
plt.title('Jobs vs. Time')
plt.ylabel('Jobs')
if Process=='Planning':
plt.xlabel('Time(days)')
else:
plt.xlabel('Time(hours)')




#plt.locator_params('both',1)
plt.yticks(range(0,max(I)+1))
plt.minorticks_on()
plt.tick_params(axis = 'both', which = 'major', labelsize = 10)
```

```
#plt.set_major_formatter(majorFormatter)


x1=0
x2=1
x3=2
x4=3
n=0
while n<len(numpylist):
x5=float(numpylist[n][x3]+numpylist[n][x4])/2
plt.text(x5,numpylist[n][x1],Identitylist[n])
n+=1

plt.show()




#print 'numpymachine',numpymachine
ganttchartm=np.array(numpymachine)

np.savetxt("ganttchartm.csv", ganttchartm, delimiter=",")
np.savetxt("numpymachine.csv", numpymachine, delimiter=",")
file = open("Identitymachine.txt", "w")
for item in Identitymachine:
file.write("%s\n" % item)
file.close()


plt.yticks(range(len(M)+1))
plt.ylim(0,len(M)+2)
plt.hlines(ganttchartm[:,2],ganttchartm[:,3],ganttchartm[:,4],
linewidth=4,colors=color_mapper(ganttchartm[:,1]))
```

```
plt.title('Machine Utilization')
plt.ylabel('Machine')
if Process=='Planning':
plt.xlabel('Time(days)')
else:
plt.xlabel('Time(hours)')


x1=0
x2=1
x3=2
x4=3
x5=4
n=0
while n<len(numpymachine):
x6=float(((numpymachine[n][x4]+numpymachine[n][x5])/2)+0.05
*numpymachine[n][x2])
plt.text(x6,numpymachine[n][x3],Identitymachine[n])
n+=1

plt.show()




#print 'numpyoperator',numpyoperator
ganttcharto=np.array(numpyoperator)

np.savetxt("ganttcharto.csv", ganttcharto, delimiter=",")
np.savetxt("numpyoperator.csv", numpyoperator, delimiter=",")
file = open("Identityoperator.txt", "w")
for item in Identityoperator:
file.write("%s\n" % item)
file.close()
```

```
#color_mapper = np.vectorize(lambda x: {1: 'red', 2: 'blue',
 3: 'green'}.get(x))
plt.yticks(range(len(O)+1))
plt.ylim(0,len(O)+2)
plt.hlines(ganttcharto[:,2],ganttcharto[:,3],ganttcharto[:,4],
linewidth=4,colors=color_mapper(ganttcharto[:,1]))
plt.title('Operator Utilization')
plt.ylabel('Operator')
if Process=='Planning':
plt.xlabel('Time(days)')
else:
plt.xlabel('Time(hours)')

x1=0
x2=1
x3=2
x4=3
x5=4
n=0
while n<len(numpyoperator):
x6=float(((numpyoperator[n][x4]+numpyoperator[n][x5])/2)+
0.05*numpyoperator[n][x2])
plt.text(x6,numpyoperator[n][x3],Identityoperator[n])
n+=1


plt.show()
```

Appendix 6 EDD-SPT Model

```
print '*DATA ARE IMPORTED...'
#earliest possible period in which project i could be
 completed
ee=[]
for i in I:
x=[]
for j in J[i]:
x.append(l[i-1][j-1])
eee=max(x)
ee.append(eee)
e=dict(zip(I,ee))


#priority of project i
w= dict(zip(I,Weight))

#time interval 1

T= dict(zip(I,(range(min(Arrival[i-1]),AbsoluteDueDate
[i-1]+1) for i in I)))

#time interval 2
T2= range(min(min(A)),max(ADD)+1)

#time interval 3
T3= {}
for i in I:
for j in J[i]: T3[i,j]= range(l[i-1][j-1],u[i-1][j-1]+1)

#time interval 4
T4={}
for i in I:
for j in J[i]: T4[i,j]= range(l[i-1][j-1],AbsoluteDue
```

```
Date[i-1]+1)

#time interval 5
T5={}
for i in I:
for j in J[i]: T5[i,j]= range(l[i-1][j-1],u[i-1][j-1]+1)

#time interval 6
TeG={}
for i in I: TeG[i]= range(e[i],AbsoluteDueDate[i-1]+1)

#time interval 7
Tal={}
for i in I:
for j in J[i]: Tal[i,j]= range(Arrival[i-1][j-1],l[i-1][j-1])

#time interval 8
TuG={}
for i in I:
for j in J[i]: TuG[i,j]= range(u[i-1][j-1]+1,Absolut
eDueDate[i-1]+1)

#time interval 9
T1e= dict(zip(I,(range(min(Arrival[i-1]),e[i]) for i in I)))

R={}
for m in M:
for t in T2: R[m,t]=1
V={}
for o in O:
for t in T2: V[o,t]=1


#desired due date for project i
```

```
g= dict(zip(I,((DesiredDueDate[i-1]) for i in I)))

Time_Start = time.clock()
print '*VARIABLES ARE GENERATED, THE MODEL IS BEING PROCESSED...
   ','
     ##',Process,'##'



#JobsToComplete
JTC=[]
for i in I:
#DurationListJob
DLJ=[]
for j in J[i]:
#DurationListStep
DLS=[]
DLS.append(i)
DLS.append(j)
DLS.append(Duration[i-1][j-1])
DLS.append(DesiredDueDate[i-1])
JTC.append(DLS)


def LESPP(i,j):
for j2 in J[i]:
if Precedence[i-1][j-1][j2-1]==1:
return 0
else: continue
return 1

LES=[]
for i in I:
for j in J[i]:
```

```
if LESPP(i,j)==1:
LES.append([i,j,Duration[i-1][j-1],DesiredDueDate[i-1]])
LES=sorted(LES,key=itemgetter(3,2))

raise
def LESP(i,j):
for j2 in J[i]:
if X[i-1][j-1][j2-1]>t:
return 0
else:
if Precedence[i-1][j-1][j2-1]==1:
return 0
else: continue
return 1


#List of eligible steps
def LESD():
del LES[:]
for i in I:
for j in J[i]:
if [i,j,Duration[i-1][j-1],DesiredDueDate[i-1]] in JTC:
if LESP(i,j)==1:
LES.append([i,j,Duration[i-1][j-1],DesiredDueDate[i-1]])
else:
continue
else:
continue



Identitylist=[]
numpylist=[]
Identitymachine=[]
```

```
numpymachine=[]
Identityoperator=[]
numpyoperator=[]

#Machine Avilability
def MA(x1,x2,t):
D=range(t,t+int(Duration[x1-1][x2-1]))
for m in MRL[x1-1][x2-1]:
Mcheck=[]
for t in D:
if (R[m,t]-Mpu[x1-1][x2-1][m-1])>=0:
Mcheck.append(1)
else:
Mcheck.append(0)
if sum(Mcheck)==len(D):
return m
else:
continue
return 0

#Operator Vavailabilty
def OA(x1,x2,t):
D=range(t,t+int(Duration[x1-1][x2-1]))
for o in ORL[x1-1][x2-1]:
Ocheck=[]
for t in D:
if (V[o,t]-Opu[x1-1][x2-1][o-1])>=0:
Ocheck.append(1)
else:
Ocheck.append(0)
if sum(Ocheck)==len(D):
return o
else:
continue
```

```
return 0

#Machine Constraint Cycle
def MCC(x1,x2,m):
if x2%2<>0:
MRL[x1-1][x2]=[]
MRL[x1-1][x2].append(m)


def AC(x1,x2,t):
if t>=Arrival[x1-1][x2-1]:
D=range(t,t+int(Duration[x1-1][x2-1]))
if MA(x1,x2,t)>=1 and OA(x1,x2,t)>=1:
m=MA(x1,x2,t)
#print "MA(X1,X2,t)",x1,x2,m
o=OA(x1,x2,t)
#print "OA(X1,X2,t)",x1,x2,o
MCC(x1,x2,m)
for t in D:
R[m,t]=R[m,t]-Mpu[x1-1][x2-1][m-1]
V[o,t]=V[o,t]-Opu[x1-1][x2-1][o-1]
for j in J[x1]:
Precedence[x1-1][j-1][x2-1]=float(0)
X[x1-1][j-1][x2-1]=int(max(D))
steplist=[]
steplist.append(x1+0.01*x2)
steplist.append(x2)
steplist.append(min(D))
steplist.append(max(D)+1)
Identitylist.append(str(x1)+"-"+str(x2))
numpylist.append(steplist)
steplist=[]
steplist.append(x1+0.01*x2)
steplist.append(x2)
```

```
steplist.append(m+0.01*x1)
steplist.append(min(D))
steplist.append(max(D)+1)
Identitymachine.append(str(x1)+"-"+str(x2))
numpymachine.append(steplist)
steplist=[]
steplist.append(x1+0.01*x2)
steplist.append(x2)
steplist.append(o+0.01*x1)
steplist.append(min(D))
steplist.append(max(D)+1)
Identityoperator.append(str(x1)+"-"+str(x2))
numpyoperator.append(steplist)

return 1



for t in T2:
if JTC:
LES=sorted(LES,key=itemgetter(3,2))
for x in LES:
x1=x[0]
x2=x[1]
if AC(x1,x2,t)==1:
JTC.remove(x)
LESD()



ganttchart=np.array(numpylist)
```

```
np.savetxt("ganttchart.csv", ganttchart, delimiter=",")
np.savetxt("numpylist.csv", numpylist, delimiter=",")
file = open("Identitylist.txt", "w")
for item in Identitylist:
file.write("%s\n" % item)
file.close()


plt.ylim(0,max(I)+1)

color= ['r', 'b', 'g', 'k', 'm', 'c', 'y']
color_mapper = np.vectorize(lambda x: {1: 'r', 2: 'b', 3: 'g', 4
:'m', 5:'c', 6:'y', 7:'r', 8:'b', 9:'g', 10:'k', 11:'m', 12:'c',
    13:'y',14:'k'}.get(x))


plt.hlines(ganttchart[:,0],ganttchart[:,2],ganttchart[:,3],
colors=color_mapper(ganttchart[:,1]),linewidth=5)
plt.title('Jobs vs. Time')
plt.ylabel('Jobs')
if Process=='Planning':
plt.xlabel('Time(days)')
else:
plt.xlabel('Time(hours)')




#plt.locator_params('both',1)
plt.yticks(range(0,max(I)+1))
plt.minorticks_on()
plt.tick_params(axis = 'both', which = 'major', labelsize = 10)
#plt.set_major_formatter(majorFormatter)
```

```
x1=0
x2=1
x3=2
x4=3
n=0
while n<len(numpylist):
x5=float(numpylist[n][x3]+numpylist[n][x4])/2
plt.text(x5,numpylist[n][x1],Identitylist[n])
n+=1

plt.show()




#print 'numpymachine',numpymachine
ganttchartm=np.array(numpymachine)

np.savetxt("ganttchartm.csv", ganttchartm, delimiter=",")
np.savetxt("numpymachine.csv", numpymachine, delimiter=",")
file = open("Identitymachine.txt", "w")
for item in Identitymachine:
file.write("%s\n" % item)
file.close()


plt.yticks(range(len(M)+1))
plt.ylim(0,len(M)+2)
plt.hlines(ganttchartm[:,2],ganttchartm[:,3],ganttchartm[:,4]
,linewidth=4,colors=color_mapper(ganttchartm[:,1]))
plt.title('Machine Utilization')
plt.ylabel('Machine')
```

```
if Process=='Planning':
plt.xlabel('Time(days)')
else:
plt.xlabel('Time(hours)')


x1=0
x2=1
x3=2
x4=3
x5=4
n=0
while n<len(numpymachine):
x6=float(((numpymachine[n][x4]+numpymachine[n][x5])/2)+0.05*
numpymachine[n][x2])
plt.text(x6,numpymachine[n][x3],Identitymachine[n])
n+=1

plt.show()




#print 'numpyoperator',numpyoperator
ganttcharto=np.array(numpyoperator)

np.savetxt("ganttcharto.csv", ganttcharto, delimiter=",")
np.savetxt("numpyoperator.csv", numpyoperator, delimiter=",")
file = open("Identityoperator.txt", "w")
for item in Identityoperator:
file.write("%s\n" % item)
file.close()

#color_mapper = np.vectorize(lambda x: {1: 'red', 2: 'blue',
```

```
3: 'green'}.get(x))
plt.yticks(range(len(O)+1))
plt.ylim(0,len(O)+2)
plt.hlines(ganttcharto[:,2],ganttcharto[:,3],ganttcharto[:,4],
linewidth=4,colors=color_mapper(ganttcharto[:,1]))
plt.title('Operator Utilization')
plt.ylabel('Operator')
if Process=='Planning':
plt.xlabel('Time(days)')
else:
plt.xlabel('Time(hours)')

x1=0
x2=1
x3=2
x4=3
x5=4
n=0
while n<len(numpyoperator):
x6=float(((numpyoperator[n][x4]+numpyoperator[n][x5])/2)+0.05*
numpyoperator[n][x2])
plt.text(x6,numpyoperator[n][x3],Identityoperator[n])
n+=1
plt.show()
```