MIXED INTEGER LINEAR PROGRAMMING APPROACH FOR SOLVING ASSEMBLY SCHEDULING PROBLEMS

BY

SHARATHRAM GANESAN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Industrial Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Adviser:

Professor Rakesh Nagi

ABSTRACT

In this research, the problem of scheduling operations in manufacturing facilities with multiple machines that produce complex multi-level assemblies is considered. The problem is modeled with precedence relationships between operations. It has been formulated as a Mixed Integer Linear Program and is modeled as a flow-like problem. The objective of the problem is to reduce the total makespan (cumulative lead time) of production of final product. In addition to that, move sizes and batch sizes of operations are taken into consideration to see their impact on total makespan. Predetermined move size is used which facilitates batch overlapping, which in turn helps reduce the makespan. Column generation method has been used to solve the problem. Subproblem generates solution sets and the master problem selects from the set of generated solutions by the subproblem. Randomly generated problems with up to 100 assembly operations, 100 batch size, and 20 workcenters are used to test the formulations. Results show that makespan increases with increase in move size. This shows that batch overlapping increases the effectiveness of a schedule.

To my family and friends, for their love and support.

ACKNOWLEDGMENTS

I would like to thank my adviser, Dr. Rakesh Nagi, for being so helpful and patient with me during this research. His help in explaining the concepts whenever needed were really necessary for me to complete this thesis. I would also like to thank Dr. Ketan Date for answering so many questions that I had in theoretical formulation and programming despite his busy schedule. I am indebted to all my friends and family for being supportive during my course of study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

JIT - Just In Time

MILP - Mixed Integer Linear Program

WIP - Work In Process

MRP - Material Requirements Planning

MRP II - Manufacturing Resource Planning

MTO - Made to Order

EOQ - Economic Order Quantity

AGV - Automated Guided Vehicle

BOM - Bill of Materials

LPR - Linear Programming Relaxation

CG - Column Generation

LP - Linear Program

CHAPTER 1

INTRODUCTION

1.1 Background

The process of scheduling is used not just in manufacturing industries but also in other areas including service industries and computer program execution. Scheduling is about allocation of resources in an efficient way by minimizing the time taken to complete the last task [1]. The scheduling function in a manufacturing and service based industry is shown in Fig 1.1 and Fig 1.2.

Scheduling deals with assignment of operations to machines and sequencing them efficiently and effectively. This implies that for a manufacturing environment to be productive, better schedules are necessary. According to Xu et al. [3], “Generating high-quality schedules has been the most difficult problem to solve”. There are different solution techniques that are used to solve the Job Shop Scheduling over these years.

With the need for more customized products, small batch manufacturing is an efficient way to manufacture. Unlike mass production, the product diversity and varying demand and due-dates make scheduling more difficult in small batch manufacturing. Customers do not want to receive large quantities of products and stock them because of inventory cost associated with them. There might be a shelf life associated with the product which might prevent from holding on to huge inventories. Just In Time (JIT) is a widely prevalent concept which encourages the practice of close to zero inventory. In addition to the inventory cost, locked capital in inventories is another reason for them to spread their demands over certain period of time. Manufacturers themselves try to keep inventory, work in process (WIP),

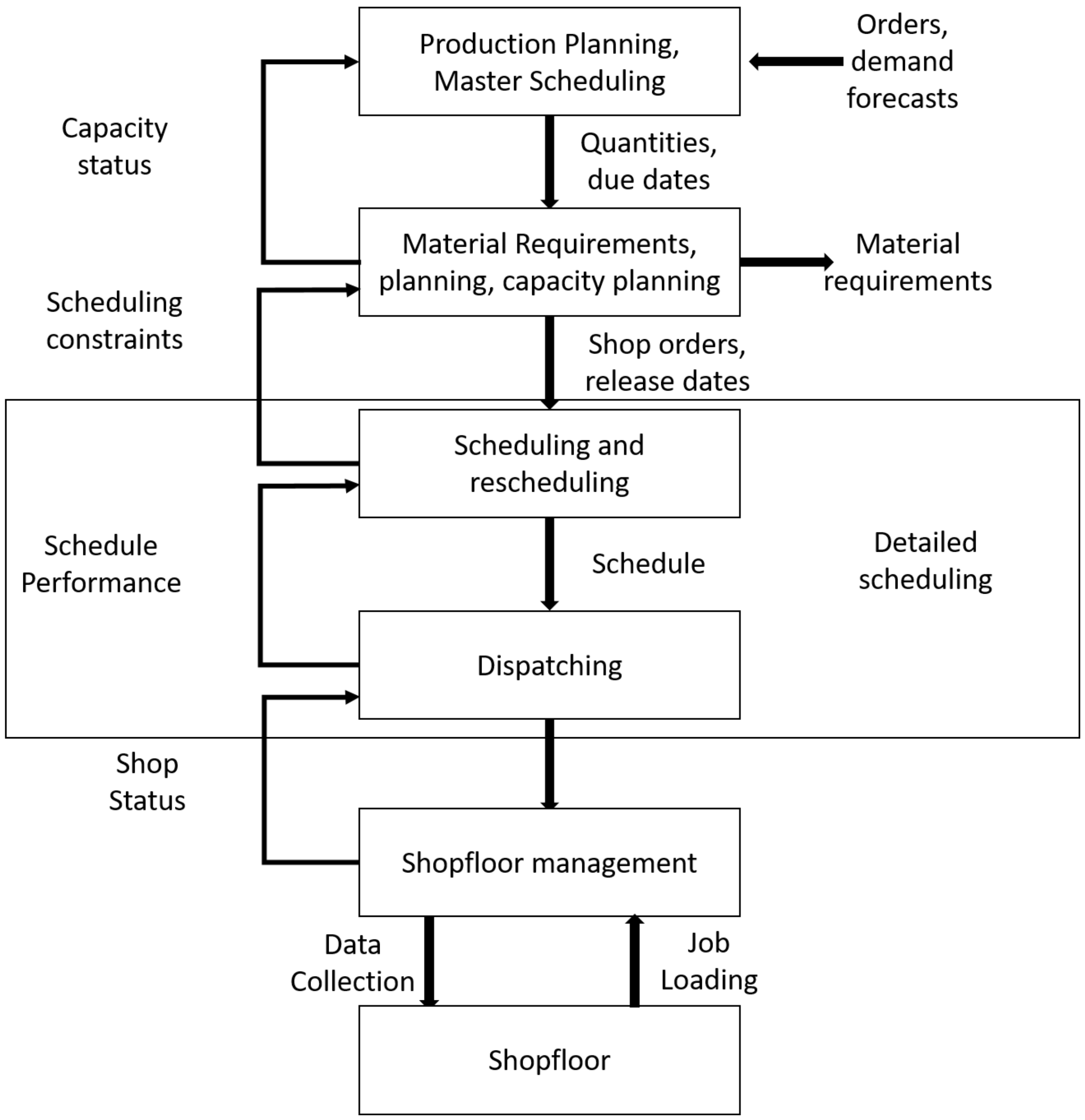


Figure 1.1: Information flow in a Service Industry [1]

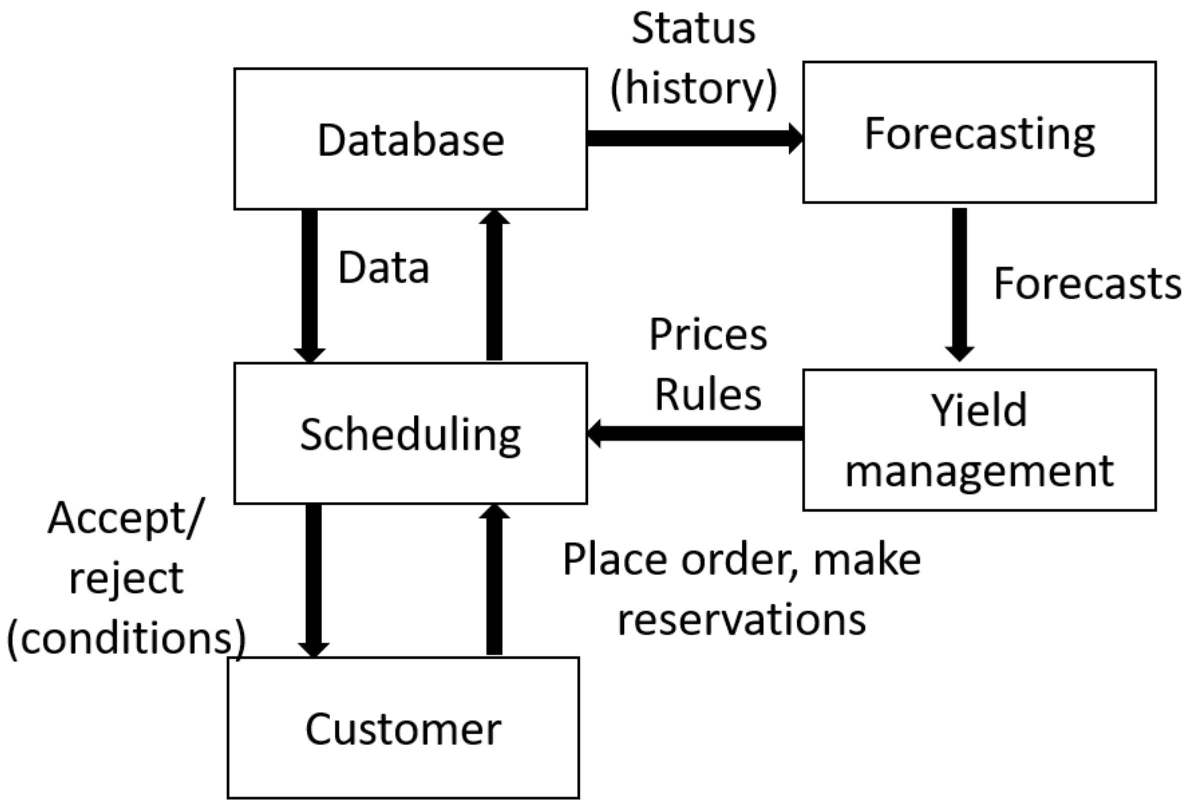


Figure 1.2: Information flow in a manufacturing facility [1]

manufacturing and transportation costs as low as possible [4].

To remain competitive in business, a manufacturing company should deliver required quantities of items on time to customers at minimal cost with the right material and specifications. The challenge in small batch manufacturing are the increasing difficulty in controlling costs due to non-value adding components. Another difficulty is meeting due dates or promised dates. Some of the non-value adding components are set-up cost, inventory holding cost and work-in-process cost. Some reasons for not being able to meet the deadline is absence of required personnel, machine breakdown/downtime, running out of materials and so on.

In order to survive the challenging market conditions, a complex assembly manufacturing facility should take efforts to plan and schedule in an efficient way. There can be many scheduling alternatives and to find the best among them, a systematic approach using operations research/heuristic is needed. Reducing makespan will save resources for the company which will eventually result in saving money. To meet the deadline, the algorithm should be able to schedule the operations in an efficient and faster way.

Plants would want to use the generated schedules promptly and hence it is not possible to solve larger problems just using a standard Mixed Integer Linear Programming (MILP) solver such as CPLEX. To support this, the problem should be solved using an effective technique which would not compromise the objective value significantly and at the same time solve it quickly. There are many methods that can be employed; one particular method that is used in this research is Column Generation. This method gives near optimal solutions for larger assembly scheduling problems and solves it faster.

1.2 Objectives

The important objective of this research is to minimize the makespan of complex assemblies while considering the resource capacity constraints. Another focus of this research is to study the effect of move sizes on batch size. To effectively solve larger problems, the objective is to use Column Generation.

This is an important issue at a complex assembly manufacturing plant as the cycle times are usually high. Overlapping batch operations by moving a partial batch to the downstream machine is an interesting way of reducing makespan and associated inventory costs at the addition of marginally more in material handling.

CHAPTER 2

LITERATURE REVIEW

For more than two machines, scheduling problems with the objective as minimizing makespan is strongly NP-hard [5]. In spite of the extensive literature on job shop scheduling, there are not many on scheduling that considers batch size and move size in the formulation.

MRP method does not consider the interdependence between stocks. While scheduling in MRP, a component of an order could be made useless because another component that is required for the same order might be late [6].

Assembly scheduling problems with tree-structured precedence constraints were considered by Xu and Nagi [3]. A complex manufacturing assembly environment with multiple workcenters and each workcenter with multiple functionally identical machines have been considered in this work. Langrangian relaxation approach with heuristics was used successfully to find near optimal solutions in less time than commercial mixed integer programming solvers [3].

Barker and McMahon describes a set of branch and bound algorithms for finding optimum schedules for job shop scheduling [7].

Agrawal et al. [2] developed a Lead-time Evaluation and Scheduling Algorithm (LETSA) to perform backward scheduling of operations in a large assembly environment with reducing cycle time as an objective. Scaled lead-time estimates were employed in an MRP based system to release work orders on the shop floor [2]. Figures 2.1 and 2.2 show the makespan and cost improvement by LETSA. Figure 2.2 shows that the cost improvement linearly increases as batch size increases.

Scheduling jobs in a manufacturing environment with identical and parallel machines have

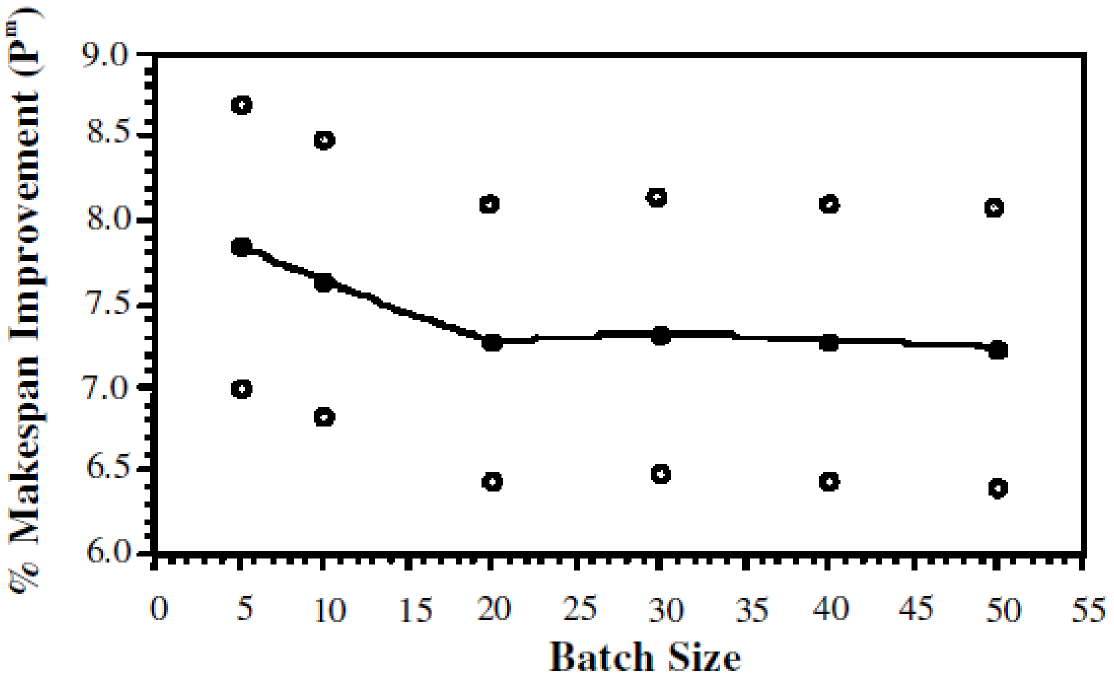


Figure 2.1: Effect of using lead time offsets on product cycle time [2]

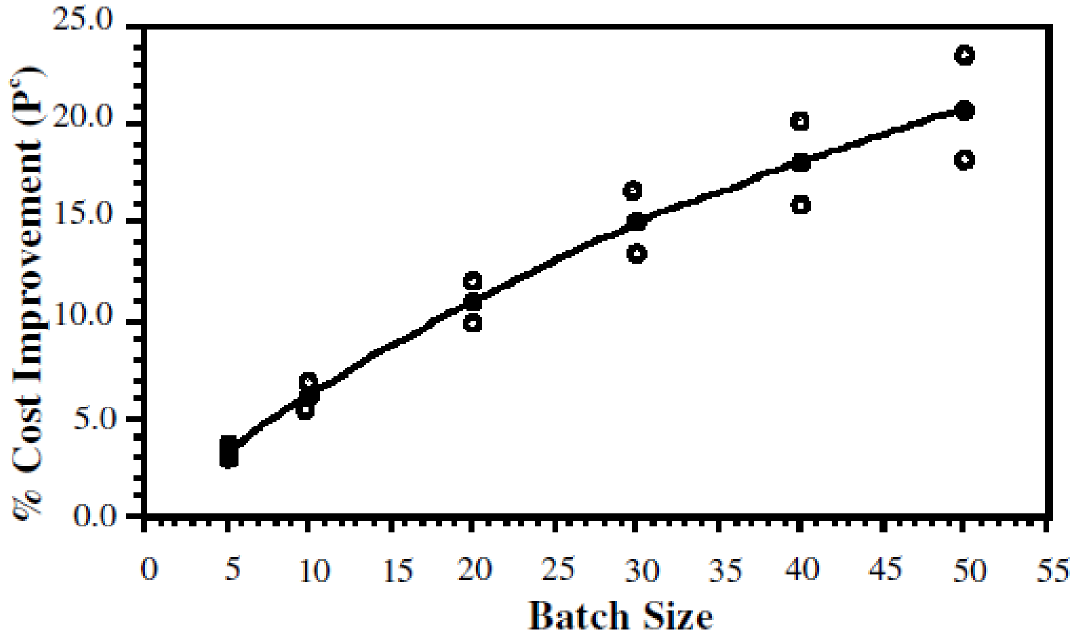


Figure 2.2: Effect of using lead time offsets on product cost [2]

been accomplished using the Lagrangian relaxation method by Hoitomt et al. [8]. Operations considered had a small precedence constraint with reducing tardiness of the total schedule as its objective [8].

For a large and complex assembly environment with a just in time production, given the due dates of end items, a scheduling problem was formulated by [4]. Lead-time evaluation and scheduling algorithm was used to prioritize the operations. Objective of their research was to minimize lead time and WIP costs [4].

Machine scheduling is applied in manufacturing, logistics, computer architecture and many other fields. Mathematical formulations for these problems are focused in this research paper [9]. This survey paper compiled a large number of mathematical programming formulations for scheduling. One of the formulations describes non-preemptive scheduling formulation for identical parallel processors (machine replicates) with minimizing makespan as the objective. This formulation also reiterates that the problem is NP-hard [9].

An integrated model for job shop scheduling and lot-sizing is presented in [10]. These problems are solved at two different levels - one where the lotsize is computed for a given sequence of jobs and another in which a sequence is calculated for given lotsizes.

Anwar and Nagi [11] considered integrated scheduling and lot-sizing problem in a complex assembly manufacturing environment. With an integrated mathematical programming, the makespan was reduced with JIT strategy to reduce the WIP. These results were compared against Lead Time Scheduling Algorithm (LETSA)[4] and the newly proposed algorithm consistently gave better results [11].

Yee and Shah [12] proposes two methods to tighten the relaxation gap. In the first technique, they focused on schedules with non-productive tasks and introduced cut functions to improve the LP schedule by adding the changeover times when solving the LP. The next one is based on a disaggregation technique, when accelerating solutions for scheduling problems with a mismatch of demands to equipment capacities [12].

Integer programs with large number of variables are discussed with their column gener-

ation solutions were discussed by Barnhart et al.
citenemhauser. Different models of problems were presented and difficulties in implementing column generation techniques were analyzed [13].

Mufalli et al. [14] used column generation to solve the sensor selection and routing of unmanned aerial vehicles. The mathematical programming model for selecting sensors and routing for UAVs simultaneously is formulated. For larger missions (larger problems) column generation found near optimal solution faster.

Dastidar et al. [15] considered multiple resource constraints with parallel machines, setup times and costs for scheduling injection molding operations. Since the computational complexity with the formulation is difficult to be solved using standard solvers, a workcenter based decomposition scheme was used to solve the problem.

Based on the literature reviews, it is found that there are scant literatures with consideration for batch overlapping. With that in consideration, a complex assembly environment with multiple workcenters and replicates where batches are processed is considered. Move size is employed to facilitate batch overlapping. Finally, a column generation method is developed to solve this problem.

CHAPTER 3

PROBLEM FORMULATION

3.1 Precedence Relationships

This research study considers a manufacturing facility with multiple workcenters, each with multiple replicates. The basic assumption is that the facility has diverse products and manufactures them in batches. There are different levels in Bill-of-Materials (BOM) and are modeled as precedence constraints.

The precedence relationship is obtained from BOM and routing sheet. A sample operation network is as shown in Figure 3.1. A sample routing information is shown in Table 3.1.

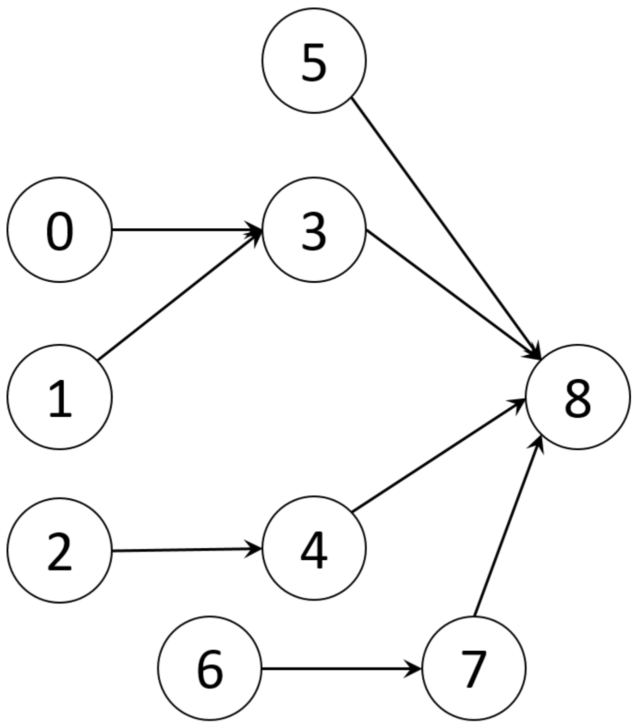


Figure 3.1: Operations Network for Final Product

Operation	Processing Time	Setup Time	Workcenter
0	4	4	1
1	3	3	2
2	10	4	2
3	8	7	1
4	6	6	2
5	3	3	2
6	3	3	1
7	6	6	2
9	10	8	2

Table 3.1: Routing Information

3.2 Problem Formulation

Let J represent the set of operations with u and v as indices. Note that the precedence of operations is known from the BOM and routings. p_u represents the processing time for operation u. K_u represents the setup time for operation u. Let D represent the due date of an end item. m denotes the move size of a batch and B denotes the batch size for an operation. In terms of variables, S_u represents start time of operation u and F_u denotes finish time of operation u.

$$x_{kuv}^{y} = \begin{cases} 1 & \text{if operation } u \text{ precedes operation } v; \qquad \forall (u,v) \in B_y, \forall k \in K_y, \forall y \in Y \\ 0 & \text{otherwise;} \end{cases}$$

Let A represent the network precedence relations $\implies A = \{(u,v) \mid u \text{ precedes } v\}$

Let Y represent the set of workcenters and each workcenter can have K_y functionally identical machines where $y \in Y$. The set of operations performed on workcenter $y \in Y$ is denoted by $J_y = \{u | u \text{ requires workcenter } y\ \}$. Let $y_u \in Y$ denote the workcenter for operation u.

Let B_y represent the set of feasible pairs of operations that are carried out on workcenter $y \implies B_y = \{(u,v)|u,v \in J_y \cup \{s,t\}; u \neq v\}$.

The set B_y can be represented as a directed network, where the nodes represent operations and arcs represent feasible operation pairs that connects two operations. s and t are dummy operations with $p_s = p_t = 0$. Each arc that enters should leave the node (node s does not have incoming arcs and node t does not have outgoing arcs).

Objective Function

$$min \;\; Z$$

subject to:

$$S_v \geq S_u + K_u + m(p_u) \qquad \text{for} \quad \{u, v \in A\} \tag{3.1}$$

$$F_v \geq F_u + m(p_v) \qquad \text{for} \quad \{u, v \in A\} \tag{3.2}$$

$$F_u = S_u + B(p_u) + K_u \qquad \text{for} \quad \{u, v \in A\} \tag{3.3}$$

$$\sum_{(u,v) \in B_y} x^y_{kuv} = 1 \qquad \text{for} \quad \{u \in J_y; y \in Y\} \tag{3.4}$$

$$\sum_{(s,v) \in B_y} x^y_{ksv} = 1 \qquad \text{for} \quad \{k \in K_y; y \in Y\} \tag{3.5}$$

$$\sum_{(u,w) \in B_y} x^y_{kuw} - \sum_{(w,v) \in B_y} x^y_{kwv} = 0 \qquad \text{for} \quad \{w \in J_y; k \in K_y; y \in Y\} \tag{3.6}$$

$$\sum_{(u,t) \in B_y} x^y_{kut} = 1 \qquad \text{for} \quad \{k \in K_y; y \in Y\} \tag{3.7}$$

$$S_u + K_u + Bp_u - M(1 - x^y_{kuv}) \leq S_v \qquad \text{for} \quad \{(u,v) \in B_y; k \in K_y; y \in Y\} \tag{3.8}$$

$$F_u \leq Z \qquad \text{for} \quad \{(u,v) \in B_y; k \in K_y; y \in Y\} \tag{3.9}$$

$$F_u \leq d_u \qquad \text{for} \quad \{(u,v) \in B_y; k \in K_y; y \in Y\} \tag{3.10}$$

$$x^y_{kuv} \in \{0,1\} \qquad \text{for} \quad \{(u,v) \in B_y; k \in K_y; y \in Y\} \tag{3.11}$$

Objective function tries to minimize the production makespan. Constraints (3.1) and (3.2) are based on network structure and they specify the relationship of start time of successor, start time of predecessor, setup time, and move size of batch and processing time of operations. Constraint (3.3) specifies the relationship between finish time and start time of operation u. Constraint (3.4) denotes that each of the operations is performed in only one of the K_y functionally identical machines. Constraints (3.5), (3.6) and (3.7) denote that any operation sequence must start at dummy operation s and end at dummy operation t. Constraint (3.8) holds the precedence relation for a workcenter. Constraint (3.9) ensures that makespan is equivalent to the largest finish time of all operations. This constraint provides the relaitonship between makespan and finish time. Constraint (3.10) makes sure that the product is produced before the due date. Constraint (3.11) indicates that x^y_{kuv} is a binary variable.

CHAPTER 4

COLUMN GENERATION

Column Generation is an efficient heuristic approach for solving Mixed Integer Linear Programming problems. This approach deals with NP hard problems by enumerating a sequence of solution columns to parts of the problem and solve an optimal collection of the alternatives from the generated sequences [16].

The column generation method used in this research has two problems - master problem and subproblem. Each subproblem generates different feasible schedules for a workcenter. On the other hand, the master problem selects a sequence from a pool of available solutions.

LP relaxation of master problem is solved and these dual variables are fed to the subproblem. Based on this, the subproblem generates sequences that are hopefully better than the LP relaxation solution of the master problem. This objective value of the relaxed master problem gives a lower bound for the optimal integer solution of the original MILP.

4.1 Master Problem

Master Problem selects set of schedules generated for a machine k of workcenter y.

Variables associated :

Ω_k^y: Set of schedules generated for machine k of workcenter y; $\forall$k $\in K_y$,$\forall$y$\in Y$;

S_u^h: Starting time of operation u in schedule h ;

$\forall h \in \Omega_k^y$, $\forall u \in J$;

X_h: Variable associated with schedule h; $\forall h \in \Omega_k^y$.

Formulation for Master Problem:

MP:

$$min \;\; Z$$

subject to:

$$Z - \sum_{h \in \Omega_k^{yu}} X_h S_u^h \geq K_u + B(p_u) \qquad \text{for} \quad \{u \in J\} \tag{4.1}$$

$$\sum_{h \in \Omega_k^{yv}} X_h S_v^h - \sum_{h \in \Omega_k^{yu}} X_h S_u^h \geq m(p_u) + K_u \qquad \text{for} \quad \{u, v \in A\} \tag{4.2}$$

$$\sum_{h \in \Omega_k^{yv}} X_h S_v^h - \sum_{h \in \Omega_k^{yu}} X_h S_u^h \geq m(p_v) + K_u + B(p_u) - K_v - B(p_v) \qquad \text{for} \quad \{u, v \in A\} \tag{4.3}$$

$$\sum_{k \in K_y} \left(\sum_{h \in \Omega_k^{yu}} X_h \left(\sum_{(u,v) \in B_y} x_{kuv}^y \right) \right) = 1 \qquad \text{for} \quad \{u \in J_y; y \in Y\} \tag{4.4}$$

$$\sum_{h \in \Omega_k^{yu}} X_h = 1 \qquad \text{for} \quad \{k \in K_y; y \in Y\} \tag{4.5}$$

$$X_h \geq 0 \quad \text{for} \;\; \{h \in \Omega_k^y; k \in K_y; y \in Y\} \tag{4.6}$$

4.2 Subproblem

SP:

$$min \; \sum_{u \in J_y} (\alpha_u) S_u + \sum_{(u,v) \in A | u \in J_y} (\beta_{uv}) S_u + \sum_{(u,v) \in A | v \in J_y} (-\beta_{uv}) S_v + \sum_{u \in J_y} (-\gamma_u^y) \left(\sum_{(u,v) \in B_y | u \in J_y} x_{kuv}^y \right) - \delta_k^y$$

subject to:

$$\sum_{(s,v)\in B_y} x^y_{ksv} = 1 \tag{4.7}$$

$$\sum_{(u,w)\in B_y} x^y_{kuw} - \sum_{(w,v)\in B_y} x^y_{kwv} = 0 \qquad \text{for} \quad \{w \in J_y\} \tag{4.8}$$

$$\sum_{(u,t)\in B_y} x^y_{kut} = 1 \tag{4.9}$$

$$S_u + Bp_u + K_u - M(1 - x^y_{kuv}) \leq S_v \qquad \text{for} \quad \{(u,v) \in B_y\} \tag{4.10}$$

$$x^y_{kuv} \in 0, 1 \qquad \text{for} \quad \{(u,v) \in B_y\} \tag{4.11}$$

$$S_u \geq 0 \qquad \text{for} \quad \{u \in J_y\} \tag{4.12}$$

CHAPTER 5

RESULTS

The MILP and column generation was tested on randomly generated problems. Formulations were coded in JAVA using Eclipse Mars on Intel Core - i5 @3.2 GHz. Cplex 12.6.1 was the optimization software used which was integrated with Eclipse Java.

An example Gantt Chart for an assembly with 10 operations is shown in Figure 5.1.

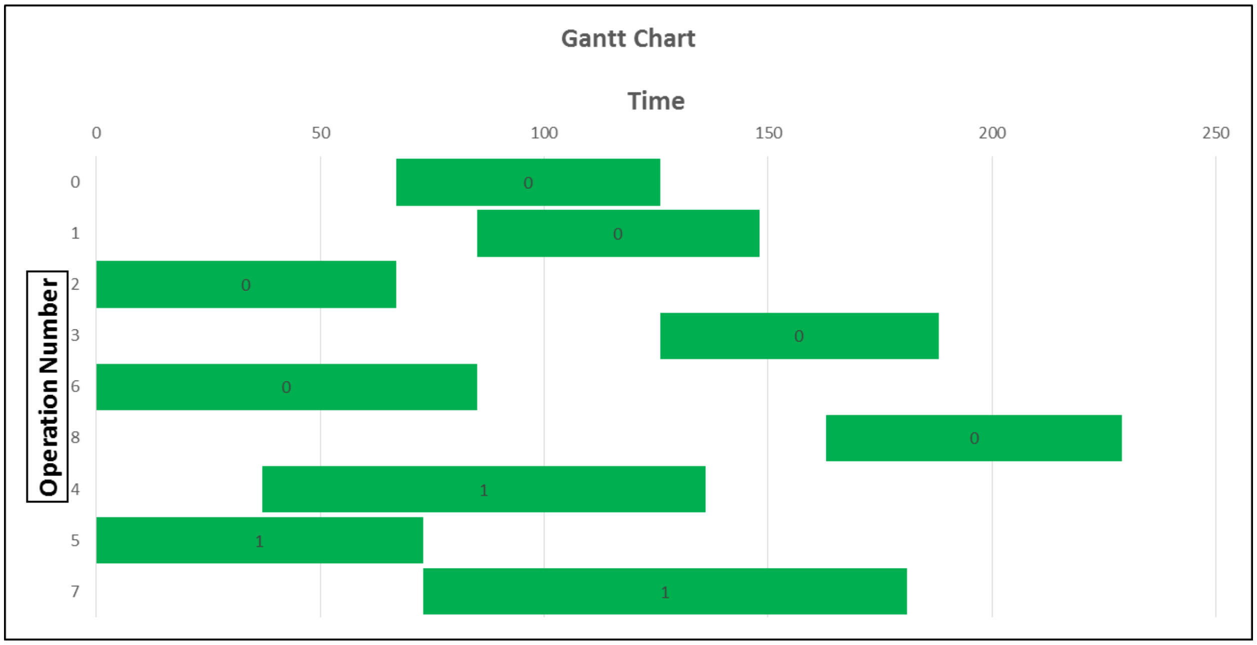
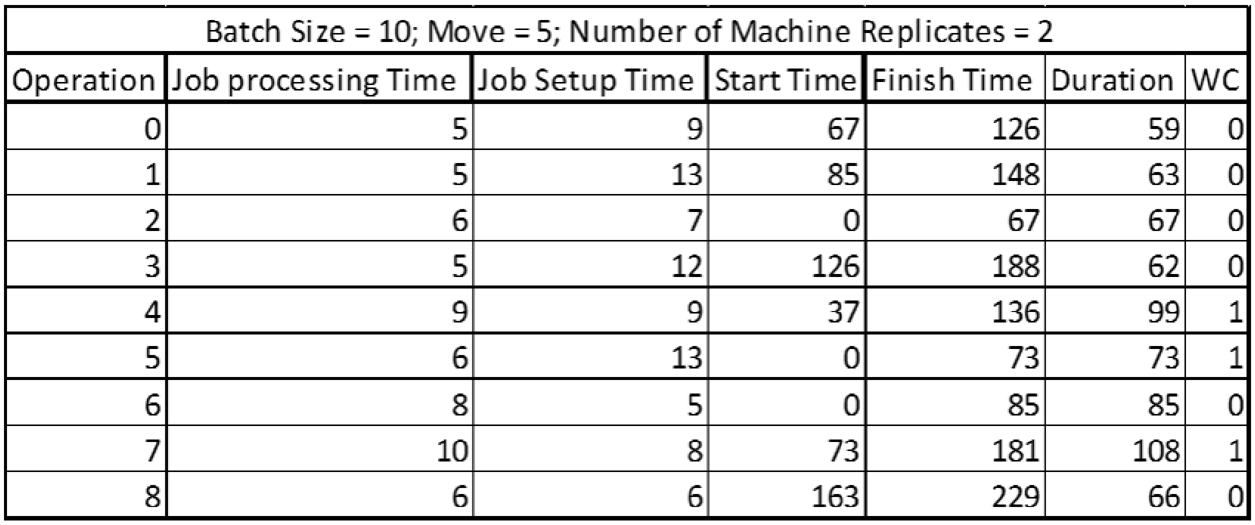


Figure 5.1: Gantt Chart for an example scheduling of 9 operations with precedence constraints as shown in the BOM Tree 3.1

5.1 Effects of move size on batch size 10

For a small batch size of 10, the effect of move sizes and performance of column generation for all move sizes are determined. The results of MILP (loosely referred to as LP) and Column



Batch Size = 10; Move = 5; Number of Machine Replicates = 2						
Operation	Job processing Time	Job Setup Time	Start Time	Finish Time	Duration	WC
0	5	9	67	126	59	0
1	5	13	85	148	63	0
2	6	7	0	67	67	0
3	5	12	126	188	62	0
4	9	9	37	136	99	1
5	6	13	0	73	73	1
6	8	5	0	85	85	0
7	10	8	73	181	108	1
8	6	6	163	229	66	0

Table 5.1: Data for Gantt Chart in Figure 5.1

Generation are shown in Table 5.2. Figure 5.2 shows that as the move size increases, there is a (nearly) linear increase in the lead time. Also, the gap between LP objective and CG solution decreases as move size increases. The percentage improvement of CG is higher for lower move size and becomes zero when move size is 10.

5.2 Effects of move size on batch size 20

For a batch size of 20, the effect of move sizes and performance of column generation for all move sizes are determined. The results of LP and Column Generation are shown in Table 5.3. Figure 5.3 shows that as move size increases, there is a nearly linear increase in the lead time. Also, the gap between LP objective and CG solution decreases as move size increases. The percentage improvement of CG is higher for lower move sizes and becomes zero when move size is 20.

5.3 Effects of move size on batch size 50

For a medium batch size of 50, the effect of move sizes and performance of column generation for all move sizes are determined. The results of LP and Column Generation are shown in

Seed	2	5	7	8
Workcenters	10	8	15	12
Number of Machines	2	3	2	2
Number of Operations	70	54	100	84
Batch size	10	10	10	10
Move Size 1 LP	208	221	219	284
CG	271.48	245.017	261.207	323.612
Move Size 2 LP	245	279	257	341
CG	297.182	279	282.828	360.405
Move Size 5 LP	356	453	371	512
CG	374.942	453	371	512
Move Size 10 LP	544	743	561	807
CG	544	743	561	807

Table 5.2: Effects of move size on batch size - 10

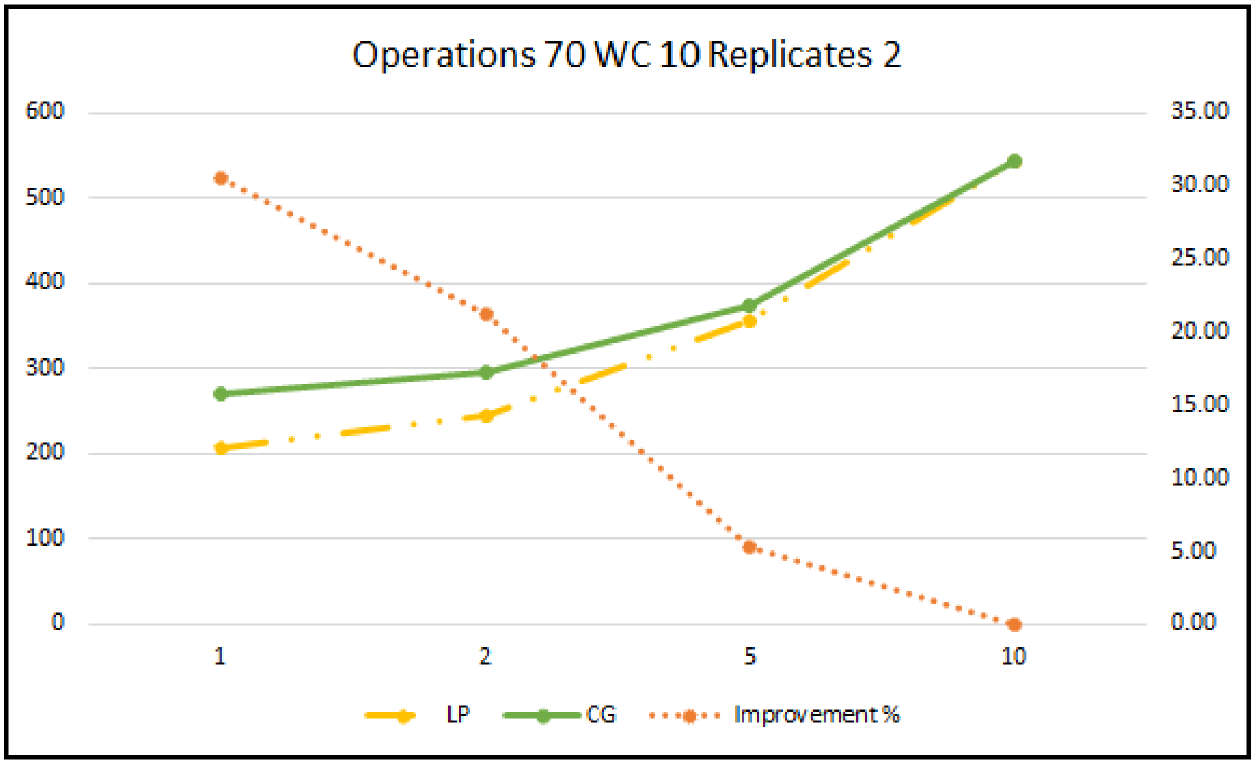


Figure 5.2: Effect of Move Size and percentage improvement of Column Generation over LP

Seed	2	5	7	8
Workcenters	10	8	15	12
Number of Machines	2	3	2	2
Number of Operations	70	54	100	84
Batch size	20	20	20	20
Move Size 1 LP	348	386	340	454
CG	478.556	452.792	465.684	555.726
Move Size 2 LP	385	414	368	511
CG	500.691	474.306	483.359	592.573
Move Size 5 LP	496	563	481	682
CG	578.424	563	544.517	703.113
Move Size 10 LP	681	853	671	977
CG	708.546	853	671.488	977
Move Size 20 LP	1051	1433	1051	1567
CG	1051	1433	1051	1567

Table 5.3: Effects of move size on batch size - 20

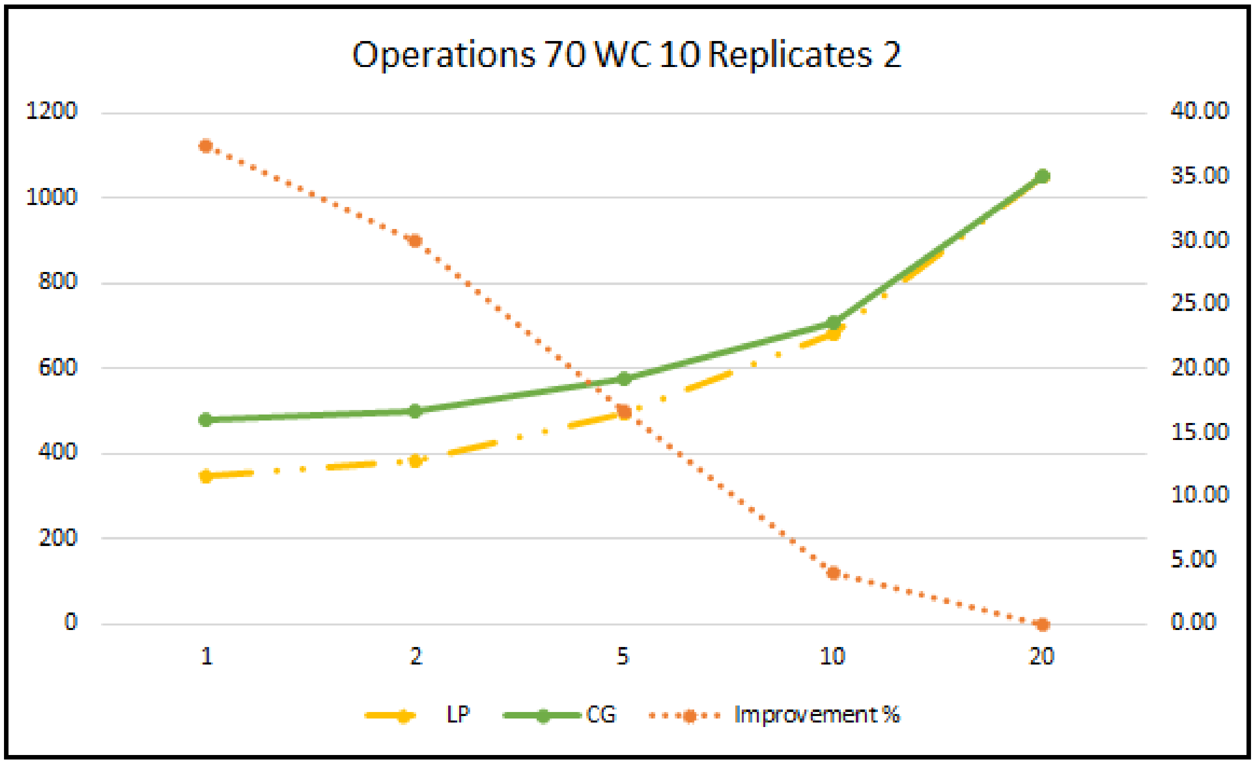


Figure 5.3: Effect of Move Size and percentage improvement of Column Generation over LP

Seed	2	5	7	8
Workcenters	10	8	15	12
Number of Machines	2	3	2	2
Number of Operations	70	54	100	84
Batch size	50	50	50	50
Move Size 1 LP	768	896	760	964
CG	1100	957.117	1079.13	1275.364
Move Size 5 LP	916	1008	872	1192
CG	1188.46	1074.17	1149.82	1399.56
Move Size 10 LP	1101	1183	1012	1487
CG	1318.29	1220.84	1238.19	1538.99
Move Size 20 LP	1471	1763	1381	2077
CG	1579.23	1763	1482.32	2077
Move Size 50 LP	2581	3503	2521	3847
CG	2581	3503	2521	3847

Table 5.4: Effects of move size on batch size - 50

Table 5.4. Figure 5.4 shows that as move size increases, there is a nearly linear increase in the lead time. Also, the gap between LP objective and CG solution decreases as move size increases. The percentage improvement of CG is higher for lower move size and becomes zero when move size is 50.

5.4 Effects of move size on batch size 100

For a large batch size of 100, the effect of move sizes and performance of column generation for all move sizes are determined. The results of LP and Column Generations are shown in Table 5.5. Figure 5.5 shows that as move size increases, there is a linear increase in the lead time. Also, the gap between LP objective and CG solution decreases as move size increases. The percentage improvement of CG is higher for lower move size and becomes zero when move size is 100.

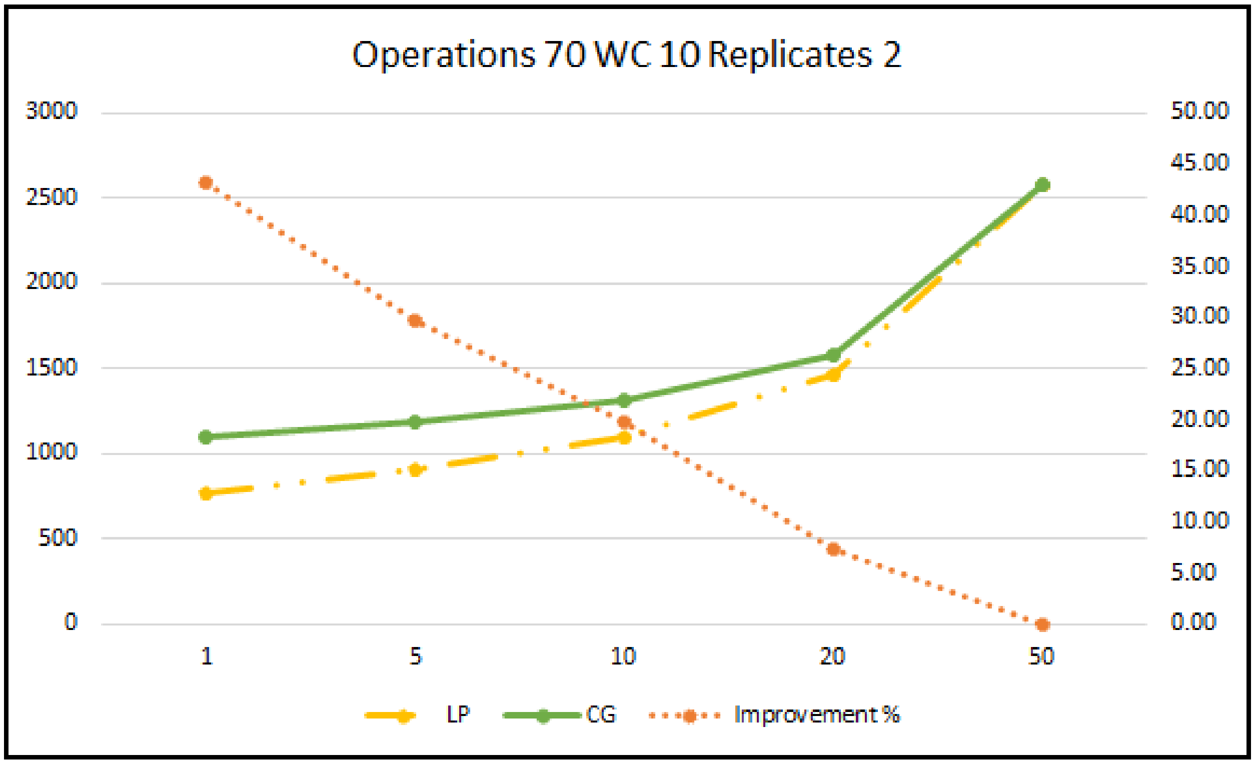


Figure 5.4: Effect of Move Size and percentage improvement of Column Generation over LP

Seed	2	5	7	8
Workcenters	10	8	15	12
Number of Machines	2	3	2	2
Number of Operations	70	54	100	84
Batch size	100	100	100	100
Move Size 1 LP	1468	1746	1460	1814
CG	2135.81	1847.63	2101.54	2525.188
Move Size 10 LP	1801	1998	1712	2337
CG	2334.78	2110.97	2260.6	2744.75
Move Size 20 LP	2171	2313	1992	2927
CG	2594.88	2404.3	2437.33	3113.76
Move Size 50 LP	3281	4053	3071	4697
CG	3378.67	4053	3172.64	4697
Move Size 100 LP	5131	6953	4971	7647
CG	5131	6953	4971	7647

Table 5.5: Effects of move size on batch size - 100

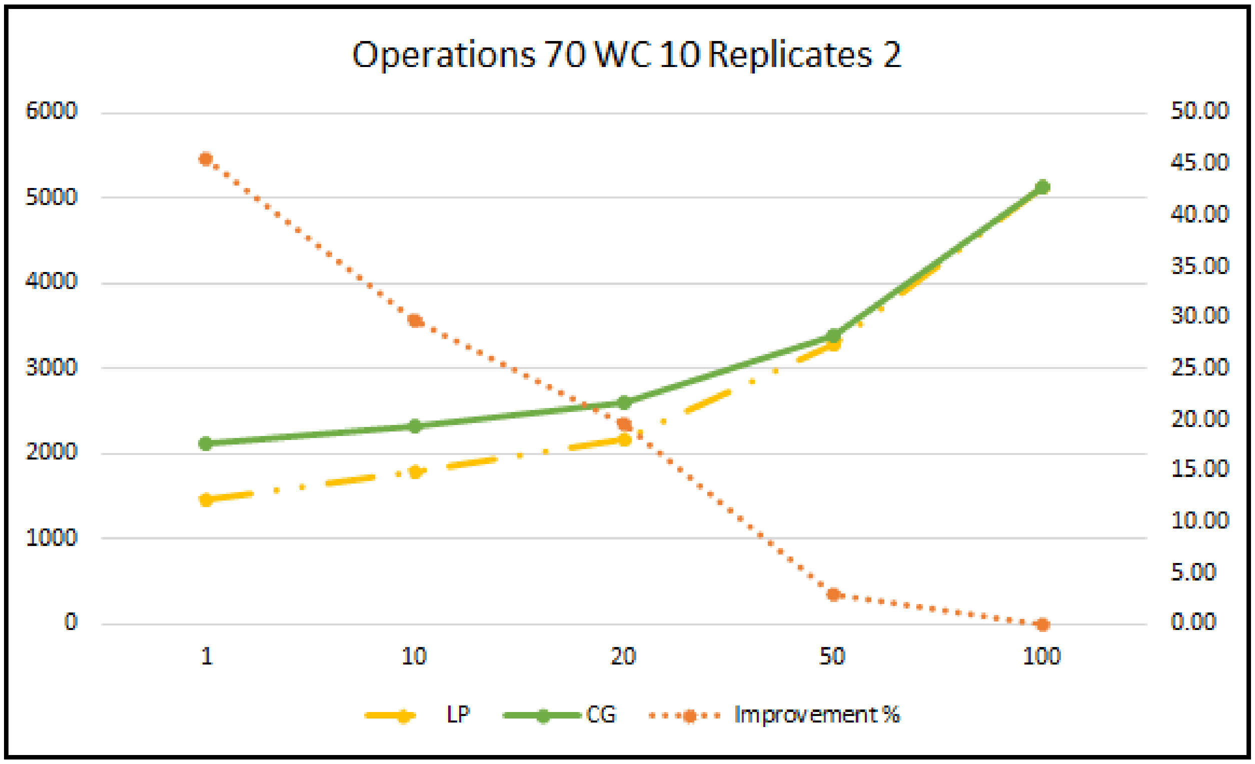


Figure 5.5: Effect of Move Size and percentage improvement of Column Generation over LP

CHAPTER 6

CONCLUSION

6.1 Conclusions

An important problem in an assembly scheduling environment is addressed in this research. Capacity constraints of machines and precedence constraints of operations have been taken into account. These are essential in a job shop scheduling environment for functioning. Batch overlap with move size consideration has been integrated in an MILP formulation. Column generation has been used to solve this problem and study the effect of move size on makespan. Multiple numerical tests were conducted to factor in many possible scenarios.

Typically, the move size reduces the lead time for the whole batch to get completed. Especially when the move sizes are small, improvement in cumulative lead time is higher. Plots shown in the Results chapter demonstrate that as move size increases, makespan increases. On the other hand, percentage improvement plot is inversely proportional to the move size.

The maximum improvement for each of the following batch sizes is shown below:

1. Batch Size 10 - 30.52
2. Batch Size 20 - 37.52
3. Batch Size 50 - 43.23
4. Batch Size 100 - 45.49

This research can be applied on many scenarios including manufacturing scheduling, software allocations, UAV/ AGV scheduling, management and more. A customer's order can be sent as partial or split orders as manufacturing is completed. This would help to maintain good relationship with customers in certain cases. Processing time can be reduced if this is used to manage multiple projects. This comes at an expense of additional logistics and material handling cost. The company should evaluate the right balance between these two metrics.

6.2 Future Work

In this research, move sizes are used as parameters (assumed to be known prior) but not as variables. Considering move size as variables increases the complexity of the problem but helps in finding the optimal value. Integrating this problem with transportation considerations between workcenters, for example, number of forklifts or AGVs to be employed in the plant and in particular workcenters can be considered. Each of those modes of transport can be assigned for few feasible routes that they can operate in. A draft of this mathematical formulation can be found in the Appendix section.

Another area that can be studied is batch-splitting. This will add more practical value because in a production environment, there might be times when a workcenter has to stopped to perform another operation before resuming the previous batch. In that batch-splitting would be an useful feature in the formulation.

CHAPTER 7

REFERENCES

[1] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 3rd ed. Springer Publishing Company, Incorporated, 2008.

[2] A. Agrawal, I. Minis, and R. Nagi, "Cycle time reduction by improved mrp-based production planning," *International Journal of Production Research*, vol. 38, no. 18, pp. 4823–4841, 2000. [Online]. Available: http://dx.doi.org/10.1080/00207540050205659

[3] J. Xu and R. Nagi, "Solving assembly scheduling problems with tree-structure precedence constraints: A lagrangian relaxation approach," *IEEE Transactions on Automation Science and Engineering*, vol. 10, no. 3, pp. 757–771, July 2013.

[4] A. Agrawal, G. Harhalakis, I. Minis, and R. Nagi, "just-in-time production of large assemblies," *IIE Transactions*, vol. 28, no. 8, pp. 653–667, 1996. [Online]. Available: http://dx.doi.org/10.1080/15458830.1996.11770710

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[6] A. Dolgui and M.-A. O. Louly, "Optimization of supply chain planning under uncertainty," in *Preprints of the IFAC Symposium on Manufacturing, Modeling, Management and Control (MIM2000)*, 2000, pp. 291–296.

[7] J. R. Barker and G. B. McMahon, "Scheduling the general job-shop," *Management Science*, vol. 31, no. 5, pp. 594–598, 1985. [Online]. Available: http://dx.doi.org/10.1287/mnsc.31.5.594

[8] D. J. Hoitomt, P. B. Luh, E. Max, and K. R. Pattipati, "Scheduling jobs with simple precedence constraints on parallel machines," *IEEE Control Systems Magazine*, vol. 10, no. 2, pp. 34–40, Feb 1990.

[9] J. Blazewicz, M. Dror, and J. Weglarz, "Mathematical programming formulations for machine scheduling: A survey," *European Journal of Operational Research*, vol. 51, no. 3, pp. 283 – 300, 1991. [Online]. Available: http://www.sciencedirect.com/science/article/pii/037722179190304E

[10] S. Dauzre-Pres and J.-B. Lasserre, "Integration of lotsizing and scheduling decisions in a job-shop," *European Journal of Operational Research*, vol. 75, no. 2, pp. 413 – 426, 1994, lotsizing models for production planning. [Online]. Available: //www.sciencedirect.com/science/article/pii/037722179490085X

[11] M. F. Anwar and R. Nagi, "Integrated lot-sizing and scheduling for just-in-time production of complex assemblies with finite set-ups," *International Journal of Production Research*, vol. 35, no. 5, pp. 1447–1470, 1997. [Online]. Available: http://dx.doi.org/10.1080/002075497195416

[12] K. Yee and N. Shah, "Improving the efficiency of discrete time scheduling formulation," *Computers & Chemical Engineering*, vol. 22, pp. S403 – S410, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0098135498000817

[13] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance, "Branch-and-price: Column generation for solving huge integer programs," *Operations Research*, vol. 46, no. 3, pp. 316–329, 1998. [Online]. Available: http://pubsonline.informs.org/doi/abs/10.1287/opre.46.3.316

[14] F. Mufalli, R. Batta, and R. Nagi, "Simultaneous sensor selection and routing of unmanned aerial vehicles for complex mission plans," *Computers & Operations Research*, vol. 39, no. 11, pp. 2787 – 2799, 2012. [Online]. Available: //www.sciencedirect.com/science/article/pii/S0305054812000366

[15] S. G. Dastidar and R. Nagi, "Scheduling injection molding operations with multiple resource constraints and sequence dependent setup times and costs," *Computers & Operations Research*, vol. 32, no. 11, pp. 2987 – 3005, 2005. [Online]. Available: //www.sciencedirect.com/science/article/pii/S0305054804000899

[16] G. B. Dantzig and P. Wolfe, "Decomposition principle for linear programs," *Oper. Res.*, vol. 8, no. 1, pp. 101–111, Feb. 1960. [Online]. Available: http://dx.doi.org/10.1287/opre.8.1.101

APPENDIX A
CPLEX/JAVA CODE FOR SOLVING MILP AND COLUMN GENERATION

Driver

```
package algorithms;
import graph.Problem;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import primitives.MachineCenter;

public class Driver {
  static final long seed = 2;
  static final String inputFile = "input.dat";
  static final String outputFile = "outlog.dat";
  static int numProblems =1;
  static int numJobs = 8;
  static int numCenters =2;
  static int numMachines = 1;
  static int batch = 10;
  static int dueDate = 5000;
  static int M = 5000;
  static int move = 1;
  static List<Problem> problemSet;
  public static void main(String[] args) {
    PrintWriter pw = null;
    generateProblems(true);
    for (int nummach = 1; nummach <= numMachines; nummach++) {
      try {
        pw = new PrintWriter(new BufferedWriter(new FileWriter("outlog"
            + nummach + ".txt")));
      } catch (IOException e) {
        e.printStackTrace();
      }
      for (int pr = 0; pr < problemSet.size(); pr++) {
        Problem PR = problemSet.get(pr);
        PR.setMachineCount(nummach);
        double[] feasibleSchedule = PR.getFeasibleSchedule(nummach);
        numJobs = PR.getJobList().size();
        numCenters = PR.getMachineList().size();
        int colLength = numJobs + PR.getJobPairList().size();
        for (int i = 0; i < PR.getJobMachineList().size(); i++) {
          colLength += PR.getJobMachineList().get(i).length;
        }
        for (MachineCenter mc : PR.getMachineList()) {
          colLength += mc.getMachineCount();
        }
        DWMaster dwm = new DWMaster();
        dwm.setNumJobs(numJobs);
        dwm.setNumCenters(numCenters);
        dwm.setColLength(colLength);
        dwm.setJobList(PR.getJobList());
        dwm.setJobPairList(PR.getJobPairList());
        dwm.setMachineList(PR.getMachineList());
        dwm.setFeasibleSchedule(feasibleSchedule);
        long start_cg = System.currentTimeMillis();
        dwm.solve();
        long end_cg = System.currentTimeMillis();
        VRPTW.setNumJobs(numJobs);
        VRPTW.setNumCenters(numCenters);
        VRPTW.setJobList(PR.getJobList());
        VRPTW.setJobPairList(PR.getJobPairList());
        VRPTW.setMachineList(PR.getMachineList());
        VRPTW.setLprelax(true);
        long start_lp = System.currentTimeMillis();
        VRPTW.main(null);
        double lpobj = VRPTW.getObjVal();
        long end_lp = System.currentTimeMillis();
        double intobj = 0;
        String st = "_" + nummach;
        String[] arr = { st };
        long start_int = System.currentTimeMillis();
        VRPTW.setLprelax(false);
        VRPTW.main(arr);
        intobj = VRPTW.getObjVal();
        long end_int = System.currentTimeMillis();
        pw.println(PR.getId() + "\t" + dwm.getUb() + "\t" +
         dwm.getObjVal() + "\t"
         + dwm.getItncount() + "\t" + dwm.getColumnSize()
          + "\t"      + (end_cg - start_cg) + "\t" + lpobj + "\t"
         + (end_lp - start_lp) + "\t" + intobj + "\t" +
         (end_int-start_int));
        pw.flush();
      }
      pw.close();
    }
  }
  private static void generateProblems(boolean readFromFile) {
    Random rng = new Random(seed);
    problemSet = new ArrayList<Problem>();
    if (readFromFile) {
      Problem pr = new Problem(0);
      pr.readProblem(inputFile, numMachines);
      problemSet.add(pr);
    } else {
      for (int p = 0; p < numProblems; p++) {
        Problem pr = new Problem(p);
        pr.generateProblem(rng, numJobs, numCenters, numMachines);
        problemSet.add(pr);
      }
    }
  }
}
```

VRPTW

```
package algorithms;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import cplex.InputDataReader;
import cplex.InputDataReader.InputDataReaderException;
import primitives.Job;
import primitives.JobPair;
import primitives.MachineCenter;
import ilog.concert.IloException;
import ilog.concert.IloIntVar;
import ilog.concert.IloLinearNumExpr;
import ilog.concert.IloNumVar;
import ilog.cplex.IloCplex;

public class VRPTW {
  private static IloCplex cplex;
  private static final String inputFile = "input.dat";
  private static final int M = Driver.M;
  private static int numJobs = Driver.numJobs;
  private static int numCenters = Driver.numCenters;
  private static final Job dummy1 = new Job(-1111, -1, 0,0);
  private static final Job dummy2 = new Job(1111, -1, 0,0);
  private static List<Job> jobList;
  private static List<JobPair> jobPairList;
  private static List<MachineCenter> machineList;
  private static List<int[]> jobMachineList;
  private static List<int[][]> jobPrecedence;
  private static IloNumVar batch;
  private static IloNumVar move;
  private static IloNumVar g_max;
  private static IloNumVar a;
  private static IloNumVar[] S;
  private static IloNumVar[][][] x;
  private static IloNumVar[] F;
  private static double objVal;
  private static boolean lprelax;
  public static void main(String[] args) {
    try {
        cplex = new IloCplex();
        objVal = 0;
        initializeDvars();
        cplex.addMinimize(g_max);
        batch = cplex.numVar(Driver.batch, Driver.batch);
        move = cplex.numVar(Driver.move, Driver.move);
        a = cplex.numVar(1, 1);
        addConstraints();
        cplex.setParam(IloCplex.DoubleParam.TiLim, 1800);
        long start = System.currentTimeMillis();
        cplex.solve();
        long end = System.currentTimeMillis();
try{
          if(lprelax){
        PrintWriter writer = new PrintWriter(new FileWriter("Start&_Finish_Time_
            LP.csv"));
        writer.printf("Operation," + "Start_Time," + "Finish_Time," + "Makespan"
            );
        writer.println();
          for (int i=0; i<numJobs; i++){
                    writer.printf(i + ","  + cplex.getValue(S[i])+ ',' + cplex.
                        getValue(F[i] )+ ','+cplex.getObjValue());
          writer.println();
          }
          writer.close();
          PrintWriter writer1 = new PrintWriter(new FileWriter ("JobList_LP.csv"
              ));
          writer1.printf("Job_Id," + "Job_Machine_ID," + "Job_processing_Time,"
              + "Job_Setup_Time," + "Pseudo_ID");
          writer1.println();
          for (Job j1 : jobList) {
          writer1.printf(j1.getJobID() + "," + j1.getMachineID() +"," + j1.
              getProcessingTime() +"," + j1.getsetupTime()+"," + j1.getPseudoID
              ());
          writer1.println();
          writer1.close();
          }
          }
          else {
            PrintWriter writer2 = new PrintWriter(new FileWriter("Start&_Finish_
                Time_IP.csv"));
            writer2.printf("Operation," + "Start_Time," + "Finish_Time," + "
                Makespan");
            writer2.println();
            for (int i=0; i<numJobs; i++){
              writer2.printf(i + ","  + cplex.getValue(S[i])+ ',' + cplex.
                  getValue(F[i] )+ ','+cplex.getObjValue());
              writer2.println();
              }
            writer2.close();
            PrintWriter writer3 = new PrintWriter(new FileWriter ("JobList_IP.
                csv"));
            writer3.printf("Job_Id," + "Job_Machine_ID," + "Job_processing_Time,
                " + "Job_Setup_Time," + "Pseudo_ID");
          writer3.println();
          for (Job j1 : jobList) {
          writer3.printf(j1.getJobID() + "," + j1.getMachineID() +"," + j1.
              getProcessingTime() +"," + j1.getsetupTime()+"," + j1.
              getPseudoID());
          writer3.println();
        }
          writer3.close();
          PrintWriter writer4 = new PrintWriter (new FileWriter ("Job␣Pair␣
              List.csv"));
          writer4.printf("Job␣1," + "Job␣1␣Machine␣ID," +  "Job␣2,"  + "Job␣2␣
              Machine␣ID");
          writer4.println();
          for ( JobPair jp : jobPairList)
          {
            writer4.printf(jp.getJ1().getJobID() + "," + jp.getJ1().
                getMachineID() + "," + jp.getJ2().getJobID() + "," + jp.getJ2
                ().getMachineID());
            writer4.println();
          }
          writer4.close();
        }
      } catch (IOException e) {
        e.printStackTrace();
      }
      if (args != null) {
        try {
          PrintWriter pw = new PrintWriter(new BufferedWriter(
              new FileWriter("gap" + args[0] + ".txt", true)));
        System.out.println("MIP␣Relative␣Gap" + cplex.getMIPRelativeGap());
          pw.println(cplex.getMIPRelativeGap());
          pw.close();
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
      cplex.exportModel("lp1.lp");
      cplex.end();
    } catch (IloException exc) {
      System.err.println("Concert␣exception␣'" + exc + "'␣caught");
    }
  }
  private static void initializeDvars() {
    try {
      g_max = cplex.numVar(0, Double.MAX_VALUE);
      S = cplex.numVarArray(numJobs, 0, Double.MAX_VALUE);
      for (int u = 0; u < numJobs; u++) {
        S[u].setName("S[" + u + "]");
      }
      F = cplex.numVarArray(numJobs,0, Double.MAX_VALUE);
      for (int u=0; u< numJobs; u++) {
        F[u].setName("F[" + u + "]");
      }
      x = new IloNumVar[numCenters][][];
```

```
        for (int y = 0; y < numCenters; y++) {
          MachineCenter mc = machineList.get(y);
          x[y] = new IloNumVar[mc.getMachineCount()][];
          for (int k = 0; k < mc.getMachineCount(); k++) {
            x[y][k] = new IloNumVar[mc.getJobPairList().size()];
            for (int uv = 0; uv < mc.getJobPairList().size(); uv++) {
              if (lprelax)
                x[y][k][uv] = cplex.numVar(0, 1);
              else
                x[y][k][uv] = cplex.boolVar();
              x[y][k][uv].setName("x[" + y + "][" + k + "][" + uv
                  + "]");
            }
          }
        }
      } catch (IloException e) {
        e.printStackTrace();
      }
    }
    private static void addConstraints() {
      try {
        for (Job j1 : jobList) {
          IloLinearNumExpr con11 = cplex.linearNumExpr();
          con11.addTerm(-1, F[j1.getJobID()]);
          con11.addTerm(1, g_max);
          cplex.addGe(con11, 0);
        }
        for (JobPair jp : jobPairList) {
          Job j1 = jp.getJ1();
          Job j2 = jp.getJ2();
          if (j1.getProcessingTime() <= j2.getProcessingTime()) {
            IloLinearNumExpr con1 = cplex.linearNumExpr();
            con1.addTerm(-1, S[j1.getJobID()]);
            con1.addTerm(1, S[j2.getJobID()]);
            con1.addTerm(-(j1.getProcessingTime()), move);
            cplex.addGe(con1, j1.getsetupTime());
        }
             else {
            IloLinearNumExpr con2 = cplex.linearNumExpr();
            con2.addTerm(1, F[j1.getJobID()]);
            con2.addTerm(move, (j2.getProcessingTime())); // m is -ve
            cplex.addLe(con2, F[j2.getJobID()]);
          }
            }
        for (Job j1 : jobList) {
          IloLinearNumExpr con8 = cplex.linearNumExpr();
          con8.addTerm(1, S[j1.getJobID()]);
          con8.addTerm(batch, j1.getProcessingTime());
          con8.addTerm(a, j1.getsetupTime());
          cplex.addEq(F[j1.getJobID()], con8);
        }
        for (MachineCenter mc : machineList) {
          int y = mc.getMachineID();
          for (Job ju : mc.getJobList()) {
```

```
        if (!(ju.equals(dummy1) || ju.equals(dummy2))) {
          IloLinearNumExpr con3 = cplex.linearNumExpr();
          for (int k = 0; k < mc.getMachineCount(); k++) {
            for (int uv = 0; uv < mc.getJobPairList().size(); uv++) {
              JobPair jp = mc.getJobPairList().get(uv);
              if (ju.equals(jp.getJ1())) {
                con3.addTerm(1, x[y][k][uv]);
              }
            }
          }
          cplex.addEq(con3, 1);
        }
      }
    }
    for (MachineCenter mc : machineList) {
      int y = mc.getMachineID();
      for (int k = 0; k < mc.getMachineCount(); k++) {
        IloLinearNumExpr con4 = cplex.linearNumExpr();
        for (int uv = 0; uv < mc.getJobPairList().size(); uv++) {
          JobPair jp = mc.getJobPairList().get(uv);
          if (jp.getJ1().equals(dummy1))
            con4.addTerm(1, x[y][k][uv]);
        }
        cplex.addEq(con4, 1);
      }
    }
    for (MachineCenter mc : machineList) {
      int y = mc.getMachineID();
      for (int k = 0; k < mc.getMachineCount(); k++) {
        IloLinearNumExpr con5 = cplex.linearNumExpr();
        for (int uv = 0; uv < mc.getJobPairList().size(); uv++) {
          JobPair jp = mc.getJobPairList().get(uv);
          if (jp.getJ2().equals(dummy2))
            con5.addTerm(1, x[y][k][uv]);
        }
        cplex.addEq(con5, 1);
      }
    }
    for (MachineCenter mc : machineList) {
      int y = mc.getMachineID();
      for (int k = 0; k < mc.getMachineCount(); k++) {
        for (Job jw : mc.getJobList()) {
          if (!(jw.equals(dummy1) || jw.equals(dummy2))) {
            IloLinearNumExpr con6 = cplex.linearNumExpr();
            for (int uv = 0; uv < mc.getJobPairList().size(); uv++) {
              JobPair jp = mc.getJobPairList().get(uv);
              if (jw.equals(jp.getJ2()))
                con6.addTerm(1, x[y][k][uv]);
              else if (jw.equals(jp.getJ1()))
                con6.addTerm(-1, x[y][k][uv]);
            }
            cplex.addEq(con6, 0);
          }
        }
```

```
          }
        }
        for (MachineCenter mc : machineList) {
          int y = mc.getMachineID();
          for (int k = 0; k < mc.getMachineCount(); k++) {
            for (int uv = 0; uv < mc.getJobPairList().size(); uv++) {
              JobPair jp = mc.getJobPairList().get(uv);
              if (!(jp.getJ1().equals(dummy1) || jp.getJ2().equals(
                  dummy2))) {
                IloLinearNumExpr con7 = cplex.linearNumExpr();
                Job j1 = jp.getJ1();
                Job j2 = jp.getJ2();
                con7.addTerm(1, S[j2.getJobID()]);
                con7.addTerm(-1, S[j1.getJobID()]);
                con7.addTerm(-M, x[y][k][uv]);
                con7.addTerm(batch, -(j1.getProcessingTime()));
                con7.addTerm(-(j1.getsetupTime()), a );
                cplex.addGe(con7, - M);
              }
            }
          }
        }
        cplex.exportModel("lp2.lp");
      } catch (IloException e) {
        e.printStackTrace();
      }
    }
    static void readData(String fileName) throws IOException,
        InputDataReader.InputDataReaderException {
      InputDataReader reader = new InputDataReader(fileName);
      jobList = new ArrayList<Job>();
      jobPairList = new ArrayList<JobPair>();
      machineList = new ArrayList<MachineCenter>();
      jobMachineList = new ArrayList<int[]>();
      jobPrecedence = new ArrayList<int[][]>();
      numJobs = reader.readInt();
      numCenters = reader.readInt();
      double[] T = reader.readDoubleArray();
      int[] M = reader.readIntArray();
      int[] P = reader.readIntArray();
      int[] K = reader.readIntArray();
      double[] set = reader.readDoubleArray();
      for (int m = 0; m < numCenters; m++) {
        int[] JM = reader.readIntArray();
        jobMachineList.add(JM);
      }
      for (int m = 0; m < numCenters; m++) {
        int[][] MP = reader.readIntArrayArray();
        jobPrecedence.add(MP);
      }
      for (int i = 0; i < numJobs; i++) {
        jobList.add(new Job(i, M[i], T[i], set [i]));
      }
      for (int i = 0; i < numJobs - 1; i++) {
        jobPairList.add(new JobPair(jobList.get(i), jobList.get(P[i])));
      }
      for (int y = 0; y < numCenters; y++) {
        machineList.add(new MachineCenter(y, K[y]));
      }
      for (MachineCenter mc : machineList) {
        int[] joblist = jobMachineList.get(mc.getMachineID());
        int[][] precedence = jobPrecedence.get(mc.getMachineID());
        mc.getJobList().add(dummy1);
        for (int i = 0; i < joblist.length; i++)
          mc.getJobList().add(jobList.get(joblist[i]));
        mc.getJobList().add(dummy2);
        for (int i = 0; i < mc.getJobList().size(); i++)
          for (int j = 0; j < mc.getJobList().size(); j++)
            if (precedence[i][j] == 1)
              mc.getJobPairList().add(
                  new JobPair(mc.getJobList().get(i), mc
                      .getJobList().get(j)));
      }
    }
    public static void setNumJobs(int numJobs) {
      VRPTW.numJobs = numJobs;
    }
    public static void setNumCenters(int numCenters) {
      VRPTW.numCenters = numCenters;
    }
    public static void setJobList(List<Job> jobList) {
      VRPTW.jobList = jobList;
    }
    public static void setJobPairList(List<JobPair> jobPairList) {
      VRPTW.jobPairList = jobPairList;
    }
    public static void setMachineList(List<MachineCenter> machineList) {
      VRPTW.machineList = machineList;
    }
    public static double getObjVal() {
      return objVal;
    }
    public static void setLprelax(boolean lprelax) {
      VRPTW.lprelax = lprelax;
    }
}
```

DWMASTER

```
package algorithms;
import graph.BOMTree;
import graph.Problem;
import ilog.concert.IloColumn;
import ilog.concert.IloException;
import ilog.concert.IloNumVar;
import ilog.concert.IloNumVarType;
import ilog.concert.IloObjective;
import ilog.concert.IloRange;
import ilog.cplex.IloCplex;
import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Random;
import primitives.Job;
import primitives.JobPair;
import primitives.MachineCenter;
import cplex.InputDataReader;

public class DWMaster {
  static int batch = Driver.batch;
  static int move = Driver.move;
  private IloCplex masterProblem;
  private List<IloNumVar> dvarArray;
  private IloRange[] conArray;
  private IloObjective mainObj;
  private static int prno = 0;
  private PrintWriter pw;
  static final Job dummyJob1 = new Job(-1111, -1, 0, 0);
  static final Job dummyJob2 = new Job(1111, -1, 0, 0);
  private int numJobs = Driver.numJobs;
  private int numCenters = Driver.numCenters;
  private int colLength = 0;
  private double[] feasibleSchedule;
  private List<Job> jobList;
  private List<JobPair> jobPairList;
  private List<MachineCenter> machineList;
  private List<int[]> jobMachineList;
  private List<DWSub> machineSubproblems;
  private int[][] constraintOffsets;
  private IloNumVar g_max;
  private IloNumVar y;
  private double dualObj;
  private double objVal;
  private double ub;
  private int columnSize;
  private int itncount;
  public void solve() {
    try {
      masterProblem = new IloCplex();
      dvarArray = new ArrayList<IloNumVar>();
      conArray = new IloRange[colLength];
      mainObj = masterProblem.addMinimize();
      dualObj = 0;
      initializeConstraints();
      initializeVariables();
      initializeReport();
      masterProblem.setOut(null);
      machineSubproblems = new ArrayList<DWSub>();
      masterProblem.setParam(IloCplex.IntParam.RootAlg,
          IloCplex.Algorithm.Primal);
      long starttime = System.currentTimeMillis();
      itncount = 0;
      objVal = 0;
      ub = 0;
      columnSize = 0;
      while (true) {
masterProblem.exportModel("lpmaster.lp");
        masterProblem.solve();
        printReport1(++itncount, starttime);
        objVal = masterProblem.getObjValue();
        System.out.println("Master Iteration: " + itncount
            + "\tTotal Columns: " + dvarArray.size()
            + "\tPrimal Obj: " + masterProblem.getObjValue()
            + "\tStatus: " + masterProblem.getStatus() +  "\tMIP Gap" +
                masterProblem.getMIPRelativeGap());
        if (solveSubproblem2() == false)
          break;
        System.out.println();
      }
      for (int i = 0; i < dvarArray.size(); i++) {
        masterProblem.add(masterProblem.conversion(dvarArray.get(i),
            IloNumVarType.Int));
      }
      masterProblem.solve();
      printReport1(++itncount, starttime);
      ub = masterProblem.getObjValue();
      columnSize = dvarArray.size();
      System.out.println();
      System.out.println("Master Iteration: " + itncount
          + "\tTotal Columns: " + dvarArray.size() + "\tPrimal Obj: "
          + masterProblem.getObjValue() + "\tStatus: "
          + masterProblem.getStatus());
      for (DWSub sp : machineSubproblems) {
        sp.end();
      }
      masterProblem.end();
      pw.close();
    } catch (IloException exc) {
      System.err.println("Concert exception '" + exc + "' caught");
    }
  }
  private boolean solveSubproblem2() {
    boolean columnsAdded = false;
    try {
      double[] dualPrices = masterProblem.getDuals(conArray);
      double[] cols = new double[dualPrices.length];
      Arrays.fill(cols, 0);
      for (MachineCenter mc : machineList) {
        double M = 0;
        for(Job jw : mc.getJobList()) {
          if(!(jw.equals(dummyJob1) || jw.equals(dummyJob2)) ) {
          M += jw.getProcessingTime();
          }
        }
        DWSub sp = new DWSub(mc, jobList, jobPairList);
        sp.setConstraintsPerBlock(constraintOffsets[mc.getMachineID()]);
        sp.setDualPrices(dualPrices);
        sp.setM(M);
        sp.solve();
        for (double[] colCoeff : sp.getColumns()) {
        IloColumn column = masterProblem.column(mainObj, 0);
          for (int p = 0; p < colCoeff.length; p++)
            column = column.and(masterProblem.column(conArray[p], colCoeff[p])
                );
          dvarArray.add(masterProblem.numVar(column, 0,Double.MAX_VALUE, ("x#"
              + dvarArray.size())));
          columnsAdded = true;
        }

        sp.end();
      }
    } catch (IloException exc) {
      System.err.println("Concert␣exception␣'" + exc + "'␣caught");
    }
    return columnsAdded;
  }
  private void initializeConstraints() {
    try {
      constraintOffsets = new int[numCenters][2];
      int offset = 0;
      for (Job j1 : jobList) {
        double tu = (batch*j1.getProcessingTime()) + j1.getsetupTime();
        conArray[offset++] = masterProblem.addRange(tu,
            Double.MAX_VALUE);
      }
      for (JobPair jp : jobPairList) {
        if(jp.getJ1().getProcessingTime() <= jp.getJ2().getProcessingTime()){
          double tu = move*jp.getJ1().getProcessingTime() + jp.getJ1().
              getsetupTime();
          conArray[offset++] = masterProblem.addRange(tu, Double.MAX_VALUE);
        } else {
          double tu = jp.getJ1().getsetupTime() + move* (jp.getJ2().
              getProcessingTime()) + batch* jp.getJ1().getProcessingTime() -
              jp.getJ2().getsetupTime() - batch* jp.getJ2().getProcessingTime
              ();
            conArray[offset++] = masterProblem.addRange(tu, Double.MAX_VALUE);
        }
          }
        for (MachineCenter mc : machineList) {
        constraintOffsets[mc.getMachineID()][0] = offset;
        for (Job jy : mc.getJobList()) {
          if (!(jy.equals(dummyJob1) || jy.equals(dummyJob2))) {
            conArray[offset++] = masterProblem.addRange(1, 1);
          }
        }
      }
      for (MachineCenter mc : machineList) {
        constraintOffsets[mc.getMachineID()][1] = offset;
        for (int k = 0; k < mc.getMachineCount(); k++) {
          conArray[offset++] = masterProblem.addRange(1, 1);
        }
      }
      } catch (IloException e) {
      System.err.println("Concert exception '" + e + "' caught");
    }
  }
  private void initializeVariables() {
    try {
      IloColumn column_g = masterProblem.column(mainObj, 1.);
      IloColumn column_y = masterProblem.column(mainObj, 0);
      int[] colCoeff_g = new int[colLength];
      double[] colCoeff_y = new double[colLength];
      Arrays.fill(colCoeff_g, 0);
      Arrays.fill(colCoeff_y, 1);
      double[] s = feasibleSchedule;
      for (int i = 0; i < numJobs; i++) {
        colCoeff_g[i] = 1;
        colCoeff_y[i] = -s[i];
      }
      int offset = numJobs;
      for (JobPair jp : jobPairList) {
        Job j1 = jp.getJ1();
        Job j2 = jp.getJ2();
        colCoeff_y[offset++] = s[j2.getJobID()] - s[j1.getJobID()];
      }
      for (int p = 0; p < colLength; p++) {
        column_y = column_y.and(masterProblem.column(conArray[p],
            colCoeff_y[p]));
      }
      for (int p = 0; p < colLength; p++) {
        column_g = column_g.and(masterProblem.column(conArray[p],
            colCoeff_g[p]));
      }
      g_max = masterProblem.numVar(column_g, 0, Double.MAX_VALUE,
          "g_max");
      dvarArray.add(g_max);
      y = masterProblem.numVar(column_y, 0, Double.MAX_VALUE, "y");
      dvarArray.add(y);
    } catch (IloException e) {
      System.err.println("Concert exception '" + e + "' caught");
    }
  }
  private void initializeReport() {
    try {
      pw = new PrintWriter(new BufferedWriter(new FileWriter("masterlog_" +
          prno + ".txt")));
      pw.println("Iteration Number\tPrimal Objective\tDual Objective\tStatus\
          tTotal Columns\tTime");
    } catch (IOException exc) {
      System.err.println("Error␣writing␣to␣file␣" + "masterlog_" + prno + ".
          txt" + ":␣"
          + exc);
    }
  }
  private void printReport1(int itncount, long starttime) {
    try {
      long current = System.currentTimeMillis();
      pw.println(itncount + "\t" + masterProblem.getObjValue() + "\t"
          + dualObj + "\t" + masterProblem.getStatus() + "\t"
          + dvarArray.size() + "\t" + (current - starttime));
      pw.flush();
    } catch (IloException e) {
      System.err.println("Concert␣exception␣'" + e + "'␣caught");
    }
  }
  public void setNumJobs(int numJobs) {
    this.numJobs = numJobs;
  }
  public void setNumCenters(int numCenters) {
    this.numCenters = numCenters;
  }
    public void setFeasibleSchedule(double[] feasibleSchedule) {
    this.feasibleSchedule = feasibleSchedule;
  }
  public void setJobList(List<Job> jobList) {
    this.jobList = jobList;
  }
  public void setJobPairList(List<JobPair> jobPairList) {
    this.jobPairList = jobPairList;
  }
  public void setJobMachineList(List<int[]> jobMachineList) {
    this.jobMachineList = jobMachineList;
  }
  public void setMachineList(List<MachineCenter> machineList) {
    this.machineList = machineList;
  }
  public void setColLength(int colLength) {
    this.colLength = colLength;
  }
  public int getItncount() {
    return itncount;
  }
  public double getObjVal() {
    return objVal;
  }
  public int getColumnSize() {
    return columnSize;
  }
  public double getUb() {
    return ub;
  }
}
```

DWSUB

```
package algorithms;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import com.sun.xml.internal.bind.v2.runtime.unmarshaller.XsiNilLoader.Array;
import ilog.concert.IloException;
import ilog.concert.IloLinearNumExpr;
import ilog.concert.IloNumVar;
import ilog.concert.IloObjective;
import ilog.cplex.IloCplex;
import primitives.Job;
import primitives.JobPair;
import primitives.MachineCenter;
import sun.swing.MenuItemLayoutHelper.ColumnAlignment;

public class DWSub {
  private static double dueDate = Driver.dueDate;
  private static int batch = Driver.batch;
  private final double RC_EPS = 1.0e-6;
  private double M = Driver.M;
  private final Job dummyJob1 = new Job(-1111, -1, 0, 0);
  private final Job dummyJob2 = new Job(1111, -1, 0, 0);
  private List<Job> jobList;
  private List<JobPair> jobPairList;
  private int totalColumns;
  private MachineCenter mc;
  private boolean[] addColumn;
  private int[] conOffsets;
  private double[] dualPrices;
  private double[] reducedCosts;
  private double[][] colCoeffs;
  private IloCplex subProblem;
  private IloObjective spObj;
  private List<double[]> columns;
  private IloNumVar[] S;
  private IloNumVar[] x;
  private double reducedCost;
  public DWSub(MachineCenter mc, List<Job> jobList, List<JobPair> jobPairList)
      {
    this.mc = mc;
    this.jobList = jobList;
    this.jobPairList = jobPairList;
    initialize();
  }
  public int solve() {
    try {
      updateObjective();
      subProblem.solve();
      generateColumns2();
      System.out.println("\tMachine Subproblem: " + getMachineID()
          + "\tColumns Added: " + totalColumns
          + "\tMin Reduced Cost: " + reducedCost + "\tStatus: "
          + subProblem.getStatus());
    } catch (IloException exc) {
      System.err.println("Concert exception '" + exc + "' caught");
    }
    return totalColumns;
  }
  private void updateObjective() {
    try {
      int index = 0;
      IloLinearNumExpr objective = subProblem.linearNumExpr();
      for (Job j : jobList) {
        if (j.getMachineID() == mc.getMachineID()) {
          objective.addTerm(dualPrices[index], S[j.getPseudoID()]);
        }
        index++;
      }
      for (JobPair jp : jobPairList) {
        Job j1 = jp.getJ1();
        Job j2 = jp.getJ2();
        if (j1.getMachineID() == mc.getMachineID()) {
          objective.addTerm(dualPrices[index], S[j1.getPseudoID()]);
        }
        if (j2.getMachineID() == mc.getMachineID()) {
          objective.addTerm(-dualPrices[index], S[j2.getPseudoID()]);
        }
        index++;
      }
      index = conOffsets[0];
      for (Job jw : mc.getJobList()) {
        if (!(jw.equals(dummyJob1) || jw.equals(dummyJob2))) {
          for (int uv = 0; uv < mc.getJobPairList().size(); uv++) {
            JobPair jp = mc.getJobPairList().get(uv);
            if (jw.equals(jp.getJ1())) {
              objective.addTerm(-dualPrices[index], x[uv]);
            }
          }
          index++;
        }
      }
      spObj.setExpr(objective);
    } catch (IloException exc) {
      System.err.println("Concert exception '" + exc + "' caught");
    }
  }
  private void generateColumns2() {
    try {
      totalColumns = 0;
      reducedCost = 0;
      double[] S_d = subProblem.getValues(S);
      double[] x_d = subProblem.getValues(x);
      int index1 = conOffsets[1];
      for (int k = 0; k < mc.getMachineCount(); k++) {
        double[] colCoeff = new double[dualPrices.length];
        Arrays.fill(colCoeff, 0);
        double reducedCost_k = subProblem.getObjValue()
            - dualPrices[index1 + k];
        if (reducedCost > reducedCost_k)
          reducedCost = reducedCost_k;
        int index = 0;
        if (reducedCost_k < -RC_EPS) {
          totalColumns++;
          for (Job j : jobList) {
            if (j.getMachineID() == mc.getMachineID()) {
              colCoeff[index] = -S_d[j.getPseudoID()];
            }
            index++;
          }
          for (JobPair jp : jobPairList) {
            Job j1 = jp.getJ1();
            Job j2 = jp.getJ2();
            if (j1.getMachineID() == mc.getMachineID()) {
              colCoeff[index] -= S_d[j1.getPseudoID()];
            }
            if (j2.getMachineID() == mc.getMachineID()) {
              colCoeff[index] += S_d[j2.getPseudoID()];
            }
            index++;
          }
          index = conOffsets[0];
          for (Job jw : mc.getJobList()) {
            if (!(jw.equals(dummyJob1) || jw.equals(dummyJob2))) {
              for (int uv = 0; uv < mc.getJobPairList().size(); uv++) {
                JobPair jp = mc.getJobPairList().get(uv);
                if (jw.equals(jp.getJ1())) {
                  colCoeff[index] += x_d[uv];
                }
              }
              index++;
            }
          }
          colCoeff[index1 + k] = 1;
          columns.add(colCoeff);
        }
      }
    } catch (IloException exc) {
      System.err.println("Concert␣exception␣'" + exc + "'␣caught");
    }
  }
  private void initialize() {
    try {
      columns = new ArrayList<double[]>();
      subProblem = new IloCplex();
      spObj = subProblem.addMinimize();
       subProblem.setOut(null);
      S = subProblem.numVarArray((mc.getJobList().size() - 2), 0,
        Double.MAX_VALUE);
    x = subProblem.boolVarArray(mc.getJobPairList().size());
    for (Job jw : mc.getJobList()) {
      if (!(jw.equals(dummyJob1) || jw.equals(dummyJob2))) {
        S[jw.getPseudoID()].setName("S[" + jw.getJobID() + "]");
      }
    }
    int index = 0;
    for (JobPair jp : mc.getJobPairList()) {
      Job j1 = jp.getJ1();
      Job j2 = jp.getJ2();
      x[index++].setName("x[" + j1.getJobID() + "][" + j2.getJobID()
          + "]");
    }
    IloLinearNumExpr con1 = subProblem.linearNumExpr();
    for (int uv = 0; uv < mc.getJobPairList().size(); uv++) {
      JobPair jp = mc.getJobPairList().get(uv);
      if (jp.getJ1().equals(dummyJob1))
        con1.addTerm(1, x[uv]);
    }
    subProblem.addEq(con1, 1);
    IloLinearNumExpr con2 = subProblem.linearNumExpr();
    for (int uv = 0; uv < mc.getJobPairList().size(); uv++) {
      JobPair jp = mc.getJobPairList().get(uv);
      if (jp.getJ2().equals(dummyJob2))
        con2.addTerm(1, x[uv]);
    }
    subProblem.addEq(con2, 1);
    for (Job jw : mc.getJobList()) {
      if (!(jw.equals(dummyJob1) || jw.equals(dummyJob2))) {
        IloLinearNumExpr con3 = subProblem.linearNumExpr();
        for (int uv = 0; uv < mc.getJobPairList().size(); uv++) {
          JobPair jp = mc.getJobPairList().get(uv);
          if (jw.equals(jp.getJ2()))
            con3.addTerm(1, x[uv]);
          else if (jw.equals(jp.getJ1()))
            con3.addTerm(-1, x[uv]);
        }
        subProblem.addEq(con3, 0);
      }
    }
    for (int uv = 0; uv < mc.getJobPairList().size(); uv++) {
      JobPair jp = mc.getJobPairList().get(uv);
      if (!(jp.getJ1().equals(dummyJob1) || jp.getJ2().equals(
          dummyJob2))) {
        IloLinearNumExpr con4 = subProblem.linearNumExpr();
        Job j1 = jp.getJ1();
        Job j2 = jp.getJ2();
        con4.addTerm(1, S[j2.getPseudoID()]);
        con4.addTerm(-1, S[j1.getPseudoID()]);
        con4.addTerm(-M, x[uv]);
         subProblem.addGe(con4, (batch * j1.getProcessingTime())- M + j1.
            getsetupTime() );
      }
      }
      for (Job jw : mc.getJobList()) {
        if (!(jw.equals(dummyJob1) || jw.equals(dummyJob2))) {
          subProblem.addLe(S[jw.getPseudoID()],(dueDate - (batch * jw.
              getProcessingTime()) - jw.getsetupTime()));
        }
      }
    } catch (IloException exc) {
      System.err.println("Concert␣exception␣'" + exc + "'␣caught");
    }
  }
  public void end() {
    subProblem.end();
  }
  public double[] getColCoeffs(int machineNo) {
    return colCoeffs[machineNo];
  }
  public boolean[] getAddColumn() {
    return addColumn;
  }
  public void setConstraintsPerBlock(int[] conOffsets) {
    this.conOffsets = conOffsets;
  }
  public void setDualPrices(double[] dualPrices) {
    this.dualPrices = dualPrices;
  }
  public double[] getReducedCosts() {
    return reducedCosts;
  }
  public int getMachineID() {
    return this.mc.getMachineID();
  }
  public int getMachineCount() {
    return this.mc.getMachineCount();
  }
  public static void setDueDate(double d) {
    DWSub.dueDate = d;
  }
  public List<double[]> getColumns() {
    return columns;
  }
  public void setM(double m) {
    M = m;
  }
}
```

INPUTDATAREADER

```
package cplex;
import java.io.*;

public class InputDataReader {
   public static class InputDataReaderException extends Exception {
      private static final long serialVersionUID = 1021L;
      InputDataReaderException(String file) {
        super("'" + file + "' contains bad data format");
    }
}
StreamTokenizer _tokenizer;
Reader _reader;
String _fileName;
public InputDataReader(String fileName) throws IOException {
    _reader = new FileReader(fileName);
    _fileName = fileName;
    _tokenizer = new StreamTokenizer(_reader);
    _tokenizer.whitespaceChars('"', '"');
    _tokenizer.whitespaceChars('\'', '\'');
    _tokenizer.ordinaryChar('[');
    _tokenizer.ordinaryChar(']');
    _tokenizer.ordinaryChar(',');
}
protected void finalize() throws Throwable {
    _reader.close();
}
double readDouble() throws InputDataReaderException,
                           IOException {
    int ntType = _tokenizer.nextToken();
    if ( ntType != StreamTokenizer.TT_NUMBER )
        throw new InputDataReaderException(_fileName);
    return _tokenizer.nval;
}
public int readInt() throws InputDataReaderException,
                     IOException {
    int ntType = _tokenizer.nextToken();
    if ( ntType != StreamTokenizer.TT_NUMBER )
        throw new InputDataReaderException(_fileName);
    return (new Double(_tokenizer.nval)).intValue();
}
public double[] readDoubleArray() throws InputDataReaderException,
                                  IOException {
    int ntType = _tokenizer.nextToken();
    if ( ntType != '[' )
        throw new InputDataReaderException(_fileName);
    DoubleArray values = new DoubleArray();
    ntType = _tokenizer.nextToken();
    while (ntType == StreamTokenizer.TT_NUMBER) {
        values.add(_tokenizer.nval);
        ntType = _tokenizer.nextToken();
        if ( ntType == ',' ) {
            ntType = _tokenizer.nextToken();
        }
        else if ( ntType != ']' ) {
            throw new InputDataReaderException(_fileName);
        }
    }
    if ( ntType != ']' )
        throw new InputDataReaderException(_fileName);
    double[] res = new double[values.getSize()];
    for (int i = 0; i < values.getSize(); i++) {
        res[i] = values.getElement(i);
    }
    return res;
}
public double[][] readDoubleArrayArray() throws InputDataReaderException,
                                           IOException {
    int ntType = _tokenizer.nextToken();
    if ( ntType != '[' )
        throw new InputDataReaderException(_fileName);
    DoubleArrayArray values = new DoubleArrayArray();
    ntType = _tokenizer.nextToken();
    while (ntType == '[') {
        _tokenizer.pushBack();
        values.add(readDoubleArray());
        ntType = _tokenizer.nextToken();
        if      ( ntType == ',' ) {
          ntType = _tokenizer.nextToken();
        }
        else if ( ntType != ']' ) {
          throw new InputDataReaderException(_fileName);
        }
    }
    if ( ntType != ']' )
        throw new InputDataReaderException(_fileName);
    double[][] res = new double[values.getSize()][];
    for (int i = 0; i < values.getSize(); i++) {
        res[i] = new double[values.getSize(i)];
        for (int j = 0; j < values.getSize(i); j++) {
            res[i][j] = values.getElement(i,j);
        }
    }
    return res;
}
public int[] readIntArray() throws InputDataReaderException,
                                   IOException {
    int ntType = _tokenizer.nextToken();
    if ( ntType != '[' )
        throw new InputDataReaderException(_fileName);
    IntArray values = new IntArray();
    ntType = _tokenizer.nextToken();
    while (ntType == StreamTokenizer.TT_NUMBER) {
        values.add(_tokenizer.nval);
        ntType = _tokenizer.nextToken();

        if      ( ntType == ',' ) {
            ntType = _tokenizer.nextToken();
        }
        else if ( ntType != ']' ) {
            throw new InputDataReaderException(_fileName);
        }
    }
    if ( ntType != ']' )
        throw new InputDataReaderException(_fileName);
    int[] res = new int[values.getSize()];
    for (int i = 0; i < values.getSize(); i++) {
        res[i] = values.getElement(i);
    }
    return res;
}
public int[][] readIntArrayArray() throws InputDataReaderException,
                                          IOException {
    int ntType = _tokenizer.nextToken();
    if ( ntType != '[' )
        throw new InputDataReaderException(_fileName);
    IntArrayArray values = new IntArrayArray();
    ntType = _tokenizer.nextToken();
    while (ntType == '[') {
        _tokenizer.pushBack();
        values.add(readIntArray());
        ntType = _tokenizer.nextToken();
        if      ( ntType == ',' ) {
            ntType = _tokenizer.nextToken();
        }
        else if ( ntType != ']' ) {
            throw new InputDataReaderException(_fileName);
        }
    }
    if ( ntType != ']' )
        throw new InputDataReaderException(_fileName);
    int[][] res = new int[values.getSize()][];
    for (int i = 0; i < values.getSize(); i++) {
        res[i] = new int[values.getSize(i)];
        for (int j = 0; j < values.getSize(i); j++) {
            res[i][j] = values.getElement(i,j);
        }
    }
    return res;
}
private class DoubleArray {
    int      _num   = 0;
    double[] _array = new double[32];
    final void add(double dval) {
        if ( _num >= _array.length ) {
            double[] array = new double[2 * _array.length];
            System.arraycopy(_array, 0, array, 0, _num);
            _array = array;
        }
        _array[_num++] = dval;
    }
    final double getElement(int i) { return _array[i]; }
    final int    getSize()         { return _num; }
}
private class DoubleArrayArray {
    int        _num   = 0;
    double[][] _array = new double[32][];
    final void add(double[] dray) {
        if ( _num >= _array.length ) {
            double[][] array = new double[2 * _array.length][];
            for (int i = 0; i < _num; i++) {
               array[i] = _array[i];
            }
            _array = array;
         }
         _array[_num] = new double[dray.length];
         System.arraycopy(dray, 0, _array[_num], 0, dray.length);
         _num++;
      }
      final double getElement(int i, int j) { return _array[i][j]; }
      final int    getSize()                { return _num; }
      final int    getSize(int i)           { return _array[i].length; }
   }
   private class IntArray {
      int   _num   = 0;
      int[] _array = new int[32];
      final void add(double ival) {
         if ( _num >= _array.length ) {
            int[] array = new int[2 * _array.length];
            System.arraycopy(_array, 0, array, 0, _num);
            _array = array;
         }
         _array[_num++] = (int)Math.round(ival);
     }
      final int getElement(int i) { return _array[i]; }
      final int getSize()         { return _num; }
   }
   private class IntArrayArray {
      int     _num   = 0;
      int[][] _array = new int[32][];
      final void add(int[] iray) {
         if ( _num >= _array.length ) {
            int[][] array = new int[2 * _array.length][];
            for (int i = 0; i < _num; i++) {
               array[i] = _array[i];
            }
            _array = array;
         }
         _array[_num] = new int[iray.length];
         System.arraycopy(iray, 0, _array[_num], 0, iray.length);
         _num++;
     }
      final int getElement(int i, int j) { return _array[i][j]; }
      final int getSize()                { return _num; }
      final int getSize(int i)           { return _array[i].length; }
   }
}
```

BOMTREE

```
package graph;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Random;
import java.util.Set;
import primitives.Job;
import primitives.JobPair;

public class BOMTree {
  private int treeID;
  private int levels;
  private List<BOMNode> elements;
  private BOMNode[] elementArray;
  private Set<BOMNode> leaves;
  private BOMNode root;
  public BOMTree(int treeID) {
    this.treeID = treeID;
    this.levels = 1;
  }
  public void populateUniform(Random rng, int numJobs) {
    elements = new ArrayList<BOMNode>();
    leaves  = new HashSet<BOMNode>();
    int nodeID = numJobs;
    root = new BOMNode(--nodeID, levels);
    elements.add(root);
    BOMNode parent = null;
    while (nodeID > 0) {
      int parentid = rng.nextInt(elements.size());
      parent = elements.get(parentid);
      int newlevel = parent.getLevel() + 1;
      if (levels < newlevel)
        levels = newlevel;
      BOMNode node = new BOMNode(--nodeID, newlevel);
      node.setParent(parent);
      parent.getChildren().add(node);
      elements.add(node);
    }
    elementArray = new BOMNode[numJobs];
    for(BOMNode node : elements) {
      elementArray[node.getNodeID()] = node;
      if(node.getChildren().isEmpty())
        leaves.add(node);
    }
  }
  public void populateFromFile(List<Job> jobList, List<JobPair> jobPairList) {
    elements = new ArrayList<BOMNode>();
    elementArray = new BOMNode[jobList.size()];
    leaves  = new HashSet<BOMNode>();
    int nodeID = jobList.size() - 1;
    root = new BOMNode(nodeID, levels);
    for(Job jw : jobList) {
      elementArray[jw.getJobID()] = new BOMNode(jw.getJobID(), levels);
    }
    for(JobPair jp : jobPairList) {
      Job j1 = jp.getJ1();
      Job j2 = jp.getJ2();
      elementArray[j1.getJobID()].setParent(elementArray[j2.getJobID()]);
      elementArray[j2.getJobID()].getChildren().add(elementArray[j1.getJobID()
          ]);
    }
    for(BOMNode node : elements) {
      if(node.getChildren().isEmpty())
        leaves.add(node);
    }
  }
  public int getTreeID() {
    return treeID;
  }
  public int getLevels() {
    return levels;
  }
  public List<BOMNode> getElements() {
    return elements;
  }
  public BOMNode getRoot() {
    return root;
  }
  public BOMNode[] getElementArray() {
    return elementArray;
  }
  public Set<BOMNode> getLeaves() {
    return leaves;
  }
}
```

BOMNODE

```
package graph;
import java.util.HashSet;
import java.util.Set;

public class BOMNode {
  private int nodeID;
  private int level;
  private BOMNode parent;
  private Set<BOMNode> children;
  public BOMNode(int nodeID, int level) {
    this.nodeID = nodeID;
    this.level = level;
    this.children = new HashSet<BOMNode>();
  }
  public int getNodeID() {
    return nodeID;
  }
  public BOMNode getParent() {
    return parent;
  }
  public void setParent(BOMNode parent) {
    this.parent = parent;
  }
  public Set<BOMNode> getChildren() {
    return children;
  }
  public int getLevel() {
    return level;
  }
  public String toString() {
    return "BOMNode [nodeID=" + nodeID + ", parent=" + (parent != null ?
        parent.getNodeID() : "null") + "]";
  }
  public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + nodeID;
    return result;
  }
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    BOMNode other = (BOMNode) obj;
    if (nodeID != other.nodeID)
      return false;
    return true;
  }
}
```

PROBLEM

```
package graph;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Random;
import java.util.Set;
import javax.naming.BinaryRefAddr;
import javax.print.attribute.standard.JobOriginatingUserName;
import algorithms.DWMaster;
import cplex.InputDataReader;
import primitives.Job;
import primitives.JobPair;
import primitives.MachineCenter;

public class Problem {
  private int id;  //newly created id
  private final Job dummyJob1 = new Job(-1111, -1, 0, 0);
  private final Job dummyJob2 = new Job(1111, -1, 0, 0);
  private List<Job> jobList;
  private List<JobPair> jobPairList;
  private List<MachineCenter> machineList;
  private List<int[]> jobMachineList;
  private List<int[][]> jobPrecedence;
  private int batch = 10;
  private double UB;
  private BOMTree bom;
  public Problem(int id) {
    this.id = id;
    bom = new BOMTree(id);
  }
  public void generateProblem(Random rng, int numJobs, int numCenters,
      int numMachines) {
    double[] T = new double[numJobs];
    int[] M = new int[numJobs];
    int[] P = new int[numJobs - 1];
    int[] K = new int[numCenters];
    double[] set = new double [numJobs];
    jobList = new ArrayList<Job>();
    jobPairList = new ArrayList<JobPair>();
    machineList = new ArrayList<MachineCenter>();
    jobMachineList = new ArrayList<int[]>();
    jobPrecedence = new ArrayList<int[][]>();
    bom.populateUniform(rng, numJobs);
    for (BOMNode node : bom.getElements()) {
      if (!node.equals(bom.getRoot())) {
        P[node.getNodeID()] = node.getParent().getNodeID();
      }
    }
    Arrays.fill(K, numMachines);
    int[] jobcount = new int[numCenters];
    Arrays.fill(jobcount, 0);
    for (int i = 0; i < T.length; i++) {
      T[i] = 5 + rng.nextInt(6);
      set[i] = 5 + rng.nextInt(10);
      int mc = rng.nextInt(numCenters);
      while(jobcount[mc] > 6) {
        mc = rng.nextInt(numCenters);
      }
      M[i] = mc;
      jobcount[mc]++;
    }
    for (int m = 0; m < numCenters; m++) {
      int[] jobarray;
      int[][] jparray;
      List<Integer> joblist = new ArrayList<Integer>();
      for (int j = 0; j < numJobs; j++) {
        if (M[j] == m) {
          joblist.add(new Integer(j));
        }
      }
      jobarray = new int[joblist.size()];
      for (int j = 0; j < jobarray.length; j++) {
        jobarray[j] = joblist.get(j).intValue();
      }
      jobMachineList.add(jobarray);
      jparray = new int[jobarray.length + 2][jobarray.length + 2];
      for (int k = 0; k < (jobarray.length + 2); k++) {
        Arrays.fill(jparray[k], 1);
        jparray[k][k] = 0;
      }
      for (int i = 0; i < jobarray.length + 2; i++)
        jparray[i][0] = 0;
      Arrays.fill(jparray[jobarray.length + 1], 0);
      for (BOMNode node : bom.getElements()) {
        int id = node.getNodeID();
        if (m == M[id]) {
          BOMNode parent = node.getParent();
            while (parent != null) {
            int pid = parent.getNodeID();
            if (m == M[pid]) {
              int id1 = Arrays.binarySearch(jobarray, id) + 1;
              int id2 = Arrays.binarySearch(jobarray, pid) + 1;
              jparray[id2][id1] = 0;
            }
            parent = parent.getParent();
          }
        }
      }
      jobPrecedence.add(jparray);
    }
    for (int i = 0; i < numJobs; i++) {
      jobList.add(new Job(i, M[i], T[i], set[i]));
    }
    for (int i = 0; i < numJobs - 1; i++) {
      jobPairList.add(new JobPair(jobList.get(i), jobList.get(P[i])));
    }
    for (int y = 0; y < numCenters; y++) {
      machineList.add(new MachineCenter(y, K[y]));
    }
    for (MachineCenter mc : machineList) {
      int[] joblist = jobMachineList.get(mc.getMachineID());
      int[][] precedence = jobPrecedence.get(mc.getMachineID());
      mc.getJobList().add(dummyJob1);
      for (int i = 0; i < joblist.length; i++) {
        jobList.get(joblist[i]).setPseudoID(i);
        mc.getJobList().add(jobList.get(joblist[i]));
      }
      mc.getJobList().add(dummyJob2);
      for (int i = 0; i < mc.getJobList().size(); i++)
        for (int j = 0; j < mc.getJobList().size(); j++)
          if (precedence[i][j] == 1)
            mc.getJobPairList().add(
                new JobPair(mc.getJobList().get(i), mc
                    .getJobList().get(j)));
    }
  }
  public void readProblem(String fileName, int numMachines) {
    try {
      InputDataReader reader = new InputDataReader(fileName);
      jobList = new ArrayList<Job>();
      jobPairList = new ArrayList<JobPair>();
      machineList = new ArrayList<MachineCenter>();
      jobMachineList = new ArrayList<int[]>();
      jobPrecedence = new ArrayList<int[][]>();
      int numJobs = reader.readInt();
      int numCenters = reader.readInt();
      double[] T = reader.readDoubleArray();
      int[] M = reader.readIntArray();
      int[] P = reader.readIntArray();
      double[] set = reader.readDoubleArray();
      for (int m = 0; m < numCenters; m++) {
        int[] JM = reader.readIntArray();
        jobMachineList.add(JM);
      }
      for (int m = 0; m < numCenters; m++) {
        int[][] MP = reader.readIntArrayArray();
        jobPrecedence.add(MP);
      }
      for (int i = 0; i < numJobs; i++) {
        jobList.add(new Job(i, M[i], T[i], set[i]));
      }
      for (int i = 0; i < numJobs - 1; i++) {
        jobPairList.add(new JobPair(jobList.get(i), jobList.get(P[i])));
      }
      for (int y = 0; y < numCenters; y++) {
        machineList.add(new MachineCenter(y, numMachines));
      }
      for (MachineCenter mc : machineList) {
        int[] joblist = jobMachineList.get(mc.getMachineID());
        int[][] precedence = jobPrecedence.get(mc.getMachineID());
        mc.getJobList().add(dummyJob1);
        for (int i = 0; i < joblist.length; i++) {
          jobList.get(joblist[i]).setPseudoID(i);
          mc.getJobList().add(jobList.get(joblist[i]));
        }
        mc.getJobList().add(dummyJob2);
        for (int i = 0; i < mc.getJobList().size(); i++)
          for (int j = 0; j < mc.getJobList().size(); j++)
            if (precedence[i][j] == 1)
              mc.getJobPairList().add(
                  new JobPair(mc.getJobList().get(i), mc.getJobList().get(j)))
                      ;
      }
      bom.populateFromFile(jobList, jobPairList);
    } catch (IOException exc) {
      System.err.println("Error␣reading␣file␣" + fileName + ":␣" + exc);
    } catch (InputDataReader.InputDataReaderException exc) {
      System.err.println(exc);
    }
  }
  public double[] getFeasibleSchedule(int numMachines) {
    UB = 0;
    double[] feasibleSchedule = new double[jobList.size()];
    double[][] machineTimes = new double[machineList.size()][numMachines];
    Arrays.fill(feasibleSchedule, 0);
  for(MachineCenter mc : machineList)
    Arrays.fill(machineTimes[mc.getMachineID()], 0);
  for (Job jw : jobList) {
    int freeID = 0;
    double freeTime = Double.MAX_VALUE;
    int jobid = jw.getJobID();
    int machineid = jw.getMachineID();
    double time = batch*jw.getProcessingTime() + jw.getsetupTime();
    for(int i = 0; i < numMachines; i++) {
      if(machineTimes[machineid][i] < freeTime) {
        freeTime = machineTimes[machineid][i];
        freeID = i;
      }
    }
    BOMNode node = bom.getElementArray()[jobid];
    if (!(node.getChildren().isEmpty())) {
      double lateststart = 0;
      for (BOMNode child : node.getChildren()) {
        int childid = child.getNodeID();
        Job jc = jobList.get(childid);
        if (lateststart < feasibleSchedule[childid] + (batch * jc.
            getProcessingTime()) + jc.getsetupTime())
          lateststart = feasibleSchedule[childid] + (batch * jc.
              getProcessingTime()) + jc.getsetupTime();
        }
      if (machineTimes[machineid][freeID] < lateststart) {
        machineTimes[machineid][freeID] = lateststart;
      }
    }
    feasibleSchedule[jobid] = machineTimes[machineid][freeID];
    machineTimes[machineid][freeID] += time;
  }
  UB = feasibleSchedule[jobList.size() - 1] + batch*(jobList.get(jobList.
      size() - 1).getProcessingTime()) + jobList.get(jobList.size() - 1).
      getsetupTime() ;
  return feasibleSchedule;
}
public int getId() {
  return id;
}
public List<int[]> getJobMachineList() {
  return jobMachineList;
}
public List<int[][]> getJoPrecedence() {
  return jobPrecedence;
}
public List<Job> getJobList() {
  return jobList;
}
public List<JobPair> getJobPairList() {
  return jobPairList;
}
public List<MachineCenter> getMachineList() {
  return machineList;
```

```
    }
    public List<int[][]> getJobPrecedence() {
        return jobPrecedence;
    }
    public double getUB() {
        return UB;
    }
    public void setMachineCount(int numMachines) {
        for(MachineCenter mc : machineList) {
            mc.setMachineCount(numMachines);
        }
    }
}
```

JOB

```
package primitives;
public class Job {
    private int jobID;
    private int machineID;
    private int pseudoID;
    private double processingTime;
    private double setupTime;
    public Job(int jobID, int machineID, double processingTime, double
        setupTime) {
        this.jobID = jobID;
        this.machineID = machineID;
        this.processingTime = processingTime;
        this.setupTime = setupTime;
    }
public int getJobID() {
        return jobID;
    }
    public void setJobID(int jobID) {
        this.jobID = jobID;
    }
    public int getMachineID() {
        return machineID;
    }
    public void setMachineID(int machineID) {
        this.machineID = machineID;
    }
    public int getPseudoID() {
        return pseudoID;
    }
    public void setPseudoID(int pseudoID) {
        this.pseudoID = pseudoID;
    }
    public double getProcessingTime() {
        return processingTime;
    }
    public void setProcessingTime(int processingTime) {
        this.processingTime = processingTime;
    }
    public double getsetupTime() {
```

```
    return setupTime;
  }
  public void setupTime(int setupTime) {
    this.setupTime = setupTime;
  }
  public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + jobID;
    return result;
  }
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    Job other = (Job) obj;
    if (jobID != other.jobID)
      return false;
    return true;
  }
}
```

JOBPAIR

```
package primitives;
public class JobPair {
  Job j1;
  Job j2;
  public JobPair(Job j1, Job j2) {
    this.j1 = j1;
    this.j2 = j2;
  }
  public Job getJ1() {
    return j1;
  }
  public void setJ1(Job j1) {
    this.j1 = j1;
  }
  public Job getJ2() {
    return j2;
  }
  public void setJ2(Job j2) {
    this.j2 = j2;
  }
  public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((j1 == null) ? 0 : j1.hashCode());
    result = prime * result + ((j2 == null) ? 0 : j2.hashCode());
    return result;
  }
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    JobPair other = (JobPair) obj;
    if (j1 == null) {
      if (other.j1 != null)
        return false;
    } else if (!j1.equals(other.j1))
      return false;
    if (j2 == null) {
      if (other.j2 != null)
        return false;
    } else if (!j2.equals(other.j2))
      return false;
    return true;
  }
}
```

MACHINECENTER

```
package primitives;
import java.util.ArrayList;
import java.util.List;
public class MachineCenter {
  private int machineID;
  private int machineCount;
  private List<Job> jobList;
  private List<JobPair> jobPairList;
  public MachineCenter(int machineID, int machineCount) {
    this.machineID = machineID;
    this.machineCount = machineCount;
    jobList = new ArrayList<Job>();
    jobPairList = new ArrayList<JobPair>();
  }
  public int getMachineID() {
    return machineID;
  }
  public void setMachineID(int machineID) {
    this.machineID = machineID;
  }
  public int getMachineCount() {
    return machineCount;
  }
  public void setMachineCount(int machineCount) {
    this.machineCount = machineCount;
  }
  public List<Job> getJobList() {
    return jobList;
  }
  public List<JobPair> getJobPairList() {
    return jobPairList;
  }
  public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + machineID;
    return result;
  }
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    MachineCenter other = (MachineCenter) obj;
    if (machineID != other.machineID)
      return false;
    return true;
  }
}
```

APPENDIX B
PROBLEM FORMULATION FOR MILP WITH AGVS

This formulation considers UAVs, forklifts or trucks, in addition to the scheduling problem.

Assumptions:

1. All the vehicles (cars) are assumed to be identical

2. One work center has many functionally identical machines

3. Each work center is capable of only 1 operation

Variables

$w(i)$ = work center of the i^{th} operation

S_{ik} = Start time of the k^{th} part of all the parallel machines of i^{th}operation.

F_{ik} = Finish time of the k^{th} part of all the parallel machines of i^{th}operation.

$$T_{i,s(i),k,l} = \begin{cases} 1 & \text{if a shipping car departs } w(i) \text{ arrives at } w(s(i)) \text{ after the } k^{th} \text{ unit of} \\ & \text{part } i \text{ is produced and arrives before the } l^{th} \text{ unit of part } s(i) \\ 0 & \text{otherwise;} \end{cases}$$

$$E_{i,s(i),k,l} = \begin{cases} 1 & \text{if an empty is sent from } w(i) \text{ to } w(s(i)) \text{, departing after the production} \\ & \text{of } k^{th} \text{ part } i \text{ and arriving before production of } l^{th} \text{ unit of part } j \\ 0 & \text{otherwise;} \end{cases}$$

$m_{i,s(i),k,l}$ = quantity of shipping from w(i) to w(s(i)) after production of output k and before the production of unit l of part j.

$I_{s(i),i,l}$ = number of part i available at $w(s(i))$ before the production of the l^{th} unit of the part j

C_{ik} = number of cars at $w(i)$ after the production of the k^{th} unit

D = Due date

n_i = number of machines in $w(i)$

K_{ik} = idle time after producing the k^{th} unit of part i

Parameters:

u_i = time needed to produce the unit part i

$q_{i,s(i)}$ = number of part i needed to produce 1 unit of part j

U_i = total number of units of the part i to be produced in a planning horizon

S = start time of the planning horizon

C_{i0} = number of cars initially at $w(i)$ after the production of 0^{th} unit

$t_{i,s(i)}$ = time taken to ship from $w(i)$ to $w(s(i))$

C_F = Fixed cost associated with each transportation

C_V = Variable cost per unit of distance traveled

C_{Ui} = variable cost per unit of the part i shipped

w = weight traded off between the make-span and the shipping cost

Objective Function:

$$Minimize : [max(D-S) + w(C_F \sum_{w(i)} \sum_{s(i)} \sum_{k=1}^{U_i} \sum_{s(k)=1}^{U_i} T_{i,s(i),k,l} + E_{i,s(i),k,l)}) \\ + C_v(\sum_{w(i)} \sum_{s(i)} \sum_{k=1}^{U_i} \sum_{l=1}^{U_i} T_{i,s(i),k,l} dist_{i,s(i)}]$$

Constraints:

$$F_{ik} + t_{w(i),w(s(i))} - S_{s(i),l} \leq (T_{i,s(i),k,l} - 1)M \qquad \forall\ l = 1, ..., U_j \tag{A.1}$$

$$0 \leq I_{s(i),i,s(k-1)} + m_{is(i)kl} - q_{i,s(i)} - I_{s(i),i,l} \leq (1 - T_{i,s(i),k,l)})M \tag{A.2}$$

$$0 \leq I_{s(i),i,l-1} + consumption_{i,s(i)} - I_{s(i),i,s(k)} \leq (T_{i,s(i),k,l})M \tag{A.3}$$

$$re0 \leq I_{iik} - I_{i,i,k-1} - n + m_{i,s(i),k,l} \leq (1 - T_{i,s(i),k,l})M \tag{A.4}$$

$$0 \leq I_{iik} - I_{,i,i,k-1} - n + m_{i,s(i),k,l} \leq (T_{i,s(i),k,l})M \tag{A.5}$$

$$C_{ik} = C_{i,k-1} + \sum_{l}^{U_{s(i)}} \sum T_{s(i),i,k,l} + \sum_{l}^{U_{s(i)}} \sum E_{s(i),i,k,l} - \sum_{l=1}^{U_{s(i)}} \sum T_{i,s(i),k,l)} - \sum_{l}^{U_{s(i)}} \sum E_{i,s(i),k,l} \tag{A.6}$$

$$D \geq F_{i,u(i)} \tag{A.7}$$

$$S_{ik} + unitprodtime_i = F_{ik} \tag{A.8}$$

$$F_{ik} + idletime_{ik} = S_{i,k+1} \tag{A.9}$$