[illegible]

Job Shop Scheduling Problem (JSSP)

Nurul Azma Abdullah

A thesis submitted to the School of Computing Sciences of the University of East Anglia in partial fulfilment of the requirements for the degree of Master of Science

Acknowledgement

First of all, I would like to thank God for giving me the possibility to complete this dissertation. I would like to express my gratitude to all those who make possible to complete my dissertation.

I am deeply indebted to my supervisor Prof. VJ Rayward Smith from the School of Computing Science University of East Anglia whose help, stimulating suggestions and encouragement helped me in all the time of studying and writing of this thesis.

I want to thank my friends in UEA, Eliana and Anati who help me in learning Latex and C++. My colleagues from Fakulti Teknologi Maklumat & Multimedia, KUITTHO supported me in my development work. I want to thank them for all their help, support, interest and valuable hints. I obliged to Zainuri for correcting my code and improve my programming skills. Not forget my friends Salvador, Eric and Vasanth for helping me in write up.

Especially, I would like to give my special thanks to my husband, Mohamad Aziz, my son, Azrul and my family who patient love and support enabled me to complete this work.

Table of Contents

Job Shop Scheduling

Nurul Azma Abdullah

October 27, 2004

Abstract

The job shop scheduling problem is a special case of the general shop in which each job i consists of n tasks Tij with processing times pij (j=1,...,m) where Tij has to be processed on machine Mij, and the tasks have to be scheduled in predetermined given order. The job shop problem is to find a feasible schedule, for example , to determine starting times for each task for all jobs to minimize the objective function of the problem.

This paper concerns in evaluate the performance of two techniques of dispatching rules which are Earliest Due Date (EDD) and Slack per Remaining Operation (SRO). The schedules are constructed based of these two heuristics. The performance of these techniques open a study on improvement techniques such as metaheuristics techniques which use a local search as a base and most recent technique introduced by David E. Joslin and David P. Clements in 1998, Squeaky Wheel which is believed to improve the schedule generated by greedy

algorithm technique.

Keywords: job shop scheduling problem, dispatching rules, earliest due date, slack per remaining operation, local search, squeaky wheel.

1 Introduction

Scheduling is the one of fundamental issue in production and manufacturing which has a significant impact on the output and performance of the organisation. Scheduling problem is the problem to assign a set of tasks to a set of resources with a consideration to a set of constraints such as deadline, resource capacities and priorities of the task. In order to provide jobs equal level of service, i.e., each job has approximately equal waiting time to be served, we need to schedule operations on machines such that their completion time variance is minimized to get optimal schedule.

There are several different methods have been developed to tackle optimality problem such as integer programming, dynamic programming, branch and bound techniques and approximation methods such as priority dispatch rules, bottlenecks based heuristics, artificial intelligence and etc. The objective of scheduling is to schedule operations so that :

1. Time consumption is less than or equal some constant, the size of the whole job should be completed.

2. Time consumption is smallest possible. = NP.

Basically, scheduling problems concentrate on open shop, job shop and flow shop scheduling. In this dissertation, the focus will be on job shop scheduling problem(JSS). The next section of the Introduction, this dissertation will focus on the nature of job shop scheduling problem and then definition of JSS problem. Various techniques to solve JSS problem will be reviewed in Literature Review section.
Then, we will prove JSS with greedy algorithm technique and then, do comparative analysis between two dispatching rules which are earliest due date (EDD) rule and Slack. The improved Slack heusristic, Slack per Remaining Operation will be discussed in future work. Before the conclusion, we will do a discussion of the improvement technique to EDD and SPO for job shop scheduling which are neighbourhood search and most recent technique, Squeaky Wheel Optimization.

2 Literature Review

The literature covering scheduling problem is quite large. The problem has been the focus of research efforts for over four decades. Job shop scheduling was proved NP-hard by Garey, Johnson and Sethi in 1976. According to [1], when job preemption is allowed, an O(n/sup 2/ log/sup 2/ n) time algorithm which can generate a minimum finish time schedule with at most min(n-2,2/sup m/-1) preemptions is obtained. When job preemption is not allowed, the problem is NP-complete. NP-complete problems are problems that are just as hard as a large number of other problems that are widely recognized as being difficult by algorithmic experts 2]. NP-complete problem can be categorized by two properties [3]:

1. No NP-complete problem is known to be solvable by a polynomial time algorithm.

2. If we had a polynomial time algorithm for one of the NP-complete problems, we could obtain polynomial time algorithms for all the NP-complete problems.

All NP complete are NP hard problem. While an NP-complete problem can be solved in polynomial time if and only if P = NP, an NP hard problem cannot be solved in polynomial time unless P = NP [4]. There are many different techniques with varying degrees of success that have been used to tackle the job shop scheduling problem. These including exact methods and approximation methods including heuristics such as neighbourhood search and metaheuristics.

2.1 Mathematical Technique

There are various techniques has been used to get an optimal schedule in job shop scheduling. Mathematical programming has been applied since four decades ago. [5] has used integer programming, then mixed integer programming to formulate the problem. [6] then used dynamic programming to tackle the problem.

After the scheduling problem was proved to be NP-complete, these approaches were seen to have limited scope in solving the scheduling problem. To tackle the problem, a group of researcher started to decompose the scheduling problem into a number of sub problems, proposing a number of techniques to solve them.

Decomposition strategies

Davis and Jones [7] proposed a methodology based on the decomposition of mathematical programming problems which used the combination of both Bender-type [8] and Dantzig/Wolfe-type [9] decompositions. Gershwin [10] proposed a mathematical programming framework for analysis of production planning and scheduling.

Enumerative techniques and Lagrangian relaxation

Branch_and_bound is an enumerative technique[11, 12]. Morton and Pentico [13] concluded, "The basic idea of branching is to conceptualize the problem as a decision tree. Each decision choice point (a node) corresponds to a partial solution. From each node, there grows a number of new branches, one for each possible decision. This branching process continues until leaf nodes, that cannot branch any further, are reached. These leaf nodes are solutions to the scheduling problem". The disadvantage of this technique is it is still very computational expensive for large scheduling problems.

Lagrangian relaxation is an old technique which has been used for more than 30 years. Lagrangian relaxation omits specific-integer constraints and adds the costs of to these omissions and/or relaxations to solve integer-programming problems. Unfortunately, this technique is also similar to branch and bound technique in that it can be computationally expensive.

2.2 Approximation methods

Approximation methods do not guarantee getting exact solution, but they are able to provide near optimal solutions, within moderate computing times. Therefore, they are more suitable for larger problems. The idea of approximation methods was supported by Glover and Greenberg (1989)[14] who suggested that directed tree searching is unsatisfactory for combinatorial problems. They stated that heuristics inspired by natural phenomena and intelligent problem solving are most suitable. In [15], four main categories of approximation technique are considered which are priority dispatching rules, bottleneck based heuristics, artificial intelligence techniques neighbourhood search and metaheuristics approach.

Dispatching rules

Dispatching rules is one of the techniques used for job shop scheduling. There are various of rules have been proposed by many researchers using this technique. These rules are designed to provide good solution to complex real time problem for specific performance criteria chose. According to Panwalker and Islander (1977) [16], the term dispatching rules is synonymous with sequencing rules, scheduling rule and heuristic.
According to Wu (1987) [17], dispatching rules can be categorized into several classes.

Class 1: contains simple priority rules, which are based on information related to the jobs. Examples of the class are shortest processing time (SPT), earliest due date (EDD), arrival time based such as first-in, first-out (FIFO) and slack based (MINSLACK).

Class 2 : a combination of rules from class one. For example, using SPT until the length exceeds 5, then switch to FIFO.

Class 3: contains rules that referred as Weight Priority Indexes which use more than one piece of information about the jobs to determine the schedule. Weight is assigned to each piece of information to reflect their importance. The jobs are ranked based on the evaluation of the objective function using the weight that has been assigned to the information.

Simulation techniques have been used to study the performance of a large number of the dispatching rules. The question asked: If you want to optimize a particular performance criterion, which rule should you choose? The early studies were concentrated on shortest processing time rule (SPT). Conway and Maxwell (1967) [18] were the pioneers in the study of the SPT rule and its variation. From the study, they found that the SPT rule minimized the mean flow time for all jobs though some job had long flow time. They also claimed that in optimizing the mean value of other basic measures such as waiting time and system utilization, SPT was the best.
The research then expanded to include of the possibility of switching rules to address a crucial problem, which is error recovery. Bean and Birge (1986) [19] and Saleh (1988) [20] have developed heuristic rules to smooth out disruptions to the original schedule. Bean and Birge adapted Turnpike Theory [21] to optimize a generalised cost function while Saleh showed that he could minimize duration of the disruption by switching the objective function from mean flow time to makespan based on disjunctive graph [22] which describe Job shop scheduling problem formally by G=(V, C U D), where

- V is a set of nodes representing operations of the jobs together with two special nodes, a source (0) and a sink *, representing the beginning and end of the schedule, respectively.
- C is a set of conjunctive arcs representing technological sequences of the operations.
- D is a set of disjunctive arcs representing pairs of operations that must be performed on the same machines.

Park [23] presented a simulation to evaluate sixteen heuristics that were adapted from Johnson's algorithm, the 'slope index' for job processing times, and minimization of total idle time in a dynamic flow-shop environment. The result showed that some of the heuristics tested perform well for the objectives selected but because of their complexity and the unrealistic assumptions used during their development, it is difficult to prove that they can perform well when applied in combination environments.

A common conclusion can been read in many studies, but originally stated by Jeremiah et al (1964) [24] is that for the makespan performance measure, no single priority rule dominates. Chang et al. (1996)[25] evaluate the performance of 42 priority dispatching rules using a linear programming model. The result showed that the SPT related rules perform consistently well while the longest processing time (LPT) related rules perform badly.

<u>Bottleneck based Heuristics</u>

According to [15], Shifting Bottleneck Procedure (SBP) is characterized by the following task:

- Subproblem identification
- Bottleneck selection
- Subproblem solution
- Schedule reoptimization

The idea involves relaxing the job shop problem into *m* one machine problems and iteratively solving each subproblem ($1|r_j|L_{max}$) one at a time by implementing the Carlier (1982) approach.

2.3 Expert/ Knowledge Based System

Expert and knowledge-based system were quite popular in the early and mid 1980s. According to [26], there are four significant advantages of them.

- They use quantitative and qualitative knowledge in the decision making.
- They manage to generate heuristics that are significantly more complex than the simple dispatching rules.

- The selection of the best heuristic can be derived based on the information about entire job shop such including the current jobs, expected new jobs, the current status of resources, material transporters and personnel.
- They capture complex relationships in new data structures and have special technique to manipulate the information in these data structures.

The disadvantages of these techniques:

- They are time consuming in building and verifying.
- They are difficult to be maintained and changed.
- Since they generate only feasible solution, it is difficult to tell how close that solution to the optimal solution.
- They can not be a generic technique

ISIS [27] was the first major expert system addressed specifically at job shop scheduling. It used a constraint-directed reasoning approach with three constraints categories which are organizational goals, physical limitations and causal restriction.

Wysk et.al (1986) [28] developed MPECS, an integrated expert system/ simulation scheduler. In order to select a small set of potentially good rules from predefined set of dispatching rules and other heuristics in the knowledge base, MPECS used both forward and backward chaining technique.

2.4 Neighbourhood search methods

Although neighbourhood search does not guarantee an exact solution it can provide near optimal solutions and offer possibilities to be enhanced when combined with other heuristics. According to Aarts and Lenstra (1997) [29], in order to generate an algorithmic solution for a given combinatorial optimisation problem with a set of feasible problem to the problem, normally we need to define configurations such as a finite set of solutions, a cost function to be optimised and a generation mechanism, which is a simple prescription to generate a transition from one configuration to another by a small change (perturbation). Method is called local search or neighbourhood search techniques. Neighbourhood search is conceptually similar to hill climbing, which these technique will continue to perturb and evaluate the solutions until there is no improvement in the objective function, which will ended the procedure.

Generally, the solution can be considered as a collection of local decisions concerning which operation to schedule next. Domdorf and Pesch, (1995) [30] suggest the construction of a framework which navigates these local decisions through the search domain in order to determine a high quality global solution in a reasonable mount of time.

Evans (1987) [31] investigates the effectiveness of local search generating mechanisms from a state space perspective. Vaessens et.al (1995) [32] presents template that captures most of the schemes proposed which suggest that multi-level local search methods need more investigation.

In 1998, Martfeld et al. [33] make an analysis of the structure of the fitness landscape of the job shop problem by experiment it with an adaptive search heuristic. The analysis indicated that adaptive search heuristics are suitable search technique for job shop problem when implement an effective navigation tool.

The other types of analysis concerning about complexity issue. Johnson et al, (1988) [34] defined the complexity class polynomial-time local search (PLS). Yannakakis (1990) [35] (1997) [36] set a formal framework regarding to the local search's complexity theory and defined the associated complexity issues more clearly.

2.5 Metaheuristics

In order to approximate search methods for solving complex optimisation problems, meta heuristics techniques have been developed [37]. Meta heuristics techniques are based on the neighbourhood strategies developed by Grabowski et al (1986, 1988) [38], Marsuo et al.(1988) [39], Van Laarhooven et al. (1992) [40] and Nowicki and Smutnicki (1996) [41].

Pirlot 1996 [42] indicates that few serious comparative studies have been performed regarding meta-solvers such as Simulated Annealing(SA), Tabu Search {TS} and Genetic Algorithms (GAs). According to his analysis GAs is the weaker method of those three in both empirical and analytical study.

Tabu Search

Glover (1989 [43], 1990[44]) define Tabu search to explore the search space of all feasible scheduling solutions by a sequence of moves. The idea is to move from one schedule to another by evaluating all candidates and choosing the best available. Some of these moves are classified as tabu, forbidden move, because they either trap the search at a local minimum, or they lead to cycling, which keep repeating part of the procedure. These moves are put onto the Tabu List, which is built up from the history of moves used during the search. These Tabu moves force exploration of the search space until the local minimum area is left behind.

Tabu search methods have been advanced by including longer term memory mechanisms. These are sometime referred as Adaptive Memory Programming (AMP)[45].

In scheduling problems, tabu search methods have been applied successfully. They have also been used as solvers of mixed integer programming problems. Nowicki and Smutnicki [45] implemented tabu search methods for job shop and flow shop scheduling problems and Vaessen [45] showed that tabu search methods for specific job shop scheduling cases perform better than other approaches such as simulated annealing, genetic algorithms and neural networks.

Greedy Randomised Adaptive Search Procedure (GRASP)

Greedy Randomised Adaptive Search Procedure (GRASP) is a problem space based method. GRASP consists of a constructive and an iterative phase. The solution to the problem is built one element at a time during the construction phase. The possible elements are candidates that can be chosen afterward. They are ordered in a restricted candidate list. (RCL). The adaptive nature of GRASP is evaluated from its ability to update the values associated with every element at each iteration, based on the selection just made. The probabilistic nature of the algorithm comes from the random selection of an element in the RCL.

During the iterative phase, the current solution is replaced with a better solution in its neighbourhood by applying local search procedure. When there is no better neighbour can be found, the phase will stop and returns to the construction phase where a new initial solution is built. The algorithms terminates when the desired solution or when a stop condition, which is a predetermined number of iterations has been met and then return the best solution so far.

Resende (1997) [46] applied an application of GRASP to job shop problem. In the application, the RCL consists of the operations that have been chosen to provide lowest overall completion time when sequenced next. The algorithm is run for more than ten million iterations on 10 SGI R10000 processors in parallel.

Simulated Annealing

Simulated annealing is based on the analogy in which liquids freeze or metal recrystalized in a cooling process of a hot metal. Initially the process starts in high temperature and is slowly cooled until its minimum energy state is reached. It is based on the proposal of Kirkpatrick et al. (1983) [47] and Cerny (1985) [48] which is a generalisation of a Monte Carlo method for examining the equations of state and frozen states of n-body systems Metropolis et al. (1953) [49]. The current state of the hot metal cooling process is analogous to the current scheduling solution, the energy equation is analogous to the objective function, and the ground state is analogous to the global minimum.

An early contribution to the neighbourhood functions for job shop problem is done by Van Laarhooven et al. (1992) [40]. The work consists moves that are achieved by reversing the processing order of an adjacent pair of critical operations. This is limited to the condition that these operations must be processed on the same machine. These neighbourhood is based on following properties:

- If x in $\mathcal{R}$ is a feasible solution, then swapping two adjacent critical operations that require the same machine can never lead to an infeasible solution
- If the swap of two adjacent non critical operations leads to a feasible solution x', then the critical path in x' cannot be shorter than the critical path in x because the critical path in x still exists in x'.
- Starting with any feasible solution x, there exists some sequence of moves that will reach an optimal solution x^* (known as the connectivity property).

Van Laarhooven et al. [40] construct a generic simulated annealing method. This method applies an annealing schedule depending on the size of the neighbourhood and bounded by m(n-1). It is proved to be robust and easy to implement and does not require a deep insight into structure of the problem.

Matsuo et al. [39] proved that if two critical adjacent operations i and j are to be swapped, the move will never be directly improving if both machine predecessors, $\mathcal{MP}(i)$ and $\mathcal{MS}(j)$ are also on the critical path. This strategy applied controlled search simulated annealing technique, a look back and lookahead strategy.

Yamada et al., 1994[50] proposed a method called Critical Block Simulated Annealing (CBSA). CBSA adapted a neighbourhood structure derived from critical blocks into SA framework. Reintensification or reannealing process is applied to improve the search. Yamada and Nakano (1995a [51], 1996a) [52] advanced CBSA with the active schedule generator by Giffler and Thompson (1960)[53] and an iterative version of SBP called Bottle Repair(BR) which is used when the neighbourhood is rejected by the Simulated Annealing.

Vakharia and Chang, 1990 [54] developed a scheduling system using simulated annealing method for manufacturing cells. Jeffcoat and Bulfin (1993)[55] applied simulated annealing to a resource-constrained scheduling problem. The results showed that the simulated annealing perform better than other neighbourhood procedures. However, Kolonko (1998) [56] stated that Simulated annealing method in job shop scheduling problem is not a convergent process and the neighbourhoods of job shop problem are not symmetric.

Genetic Algorithms

Genetic algorithms were formally introduced in the United States in the 1970s by John Holland at University of Michigan. It is an optimization methodology which is based on analogy to Darwinian natural selection and mutation in biological reproduction. The continuing price/performance improvements of computational systems have attracted AI researchers for some types of optimization. Genetic algorithms work very well on mixed (continuous and discrete), combinatorial problems. They have less chance to getting trapped at local optima compared with other search methods but at a higher price they tend to be computationally expensive.

To use a genetic algorithm, a solution to the problem has to be represented as a *genome* (or *chromosome*). Genetic Algorithms use genetic operators such as mutation and crossover to evolve the solutions in order to find the best one.

A number of approaches have been applied genetic algorithm to job shop scheduling problems Davis 1985 [57], Goldberg and Lingle 1985 [58]:

- Job shop scheduling using genetic algorithms with blind recombination operators.
- The solution to sequencing problems by mapping their constraints to a Boolean satisfiability problem using partial payoff schemes
- Job shop scheduling with heuristic genetic algorithms.

2.6 Conclusion

Prior contributions in job shop scheduling problem are very extensive. This dissertation seeks to compare two rules in dispatching rules technique, which are earliest due date (EDD) rule and slack per remaining operation (SPO). The discussion to improve the performance of both rules will focus on Neighbourhood search and a new technique known as Squeaky Wheel.

3 Job Shop Scheduling

Job shop scheduling (JSS) is an NP-hard problem[59]. Thus, in general, it is highly unlikely there exists a polynomial time algorithm to find the optimal schedule.

For particular cases, however, polynomial time algorithms do exist. The first case is JSS with two machines and at most two operations per job and the second case is JSS with two jobs[60]. These two cases can be solved with polynomial time algorithm. Except got the above cases, JSS is NP-hard problem. Even with fast unit processing time, JSS with just 3 machines remains NP-hard.

3.1 Term in Job Shop Scheduling Problem

Makespan

The completion time of the last job to leave the system. A minimum makespan usually implies a high utilization of the machines.

Non-preemptive

Implies that it is not possible to interrupt a job on a machine until completion.

Processing Time (P_{ki})

P_{ki} represents the processing time of job i on machine k.

Release Date (r_i)

The release date, r_i of job i may also be referred to as the ready date, the date when the job arrives at the system, which is the earliest time job i can start its processing.

Due date(d_i)

The due date of job i, is the completion date of job i. The completion time of a job after due date is allowed, but penalty is incurred. It is referred as deadline when it is absolutely must be met.

3.2 Problem Definition

We are given a set of m machines,

$$M = m_1, m_2, ..., m_m$$

and a set of n jobs

$$J = \{j_1, j_2, j_3, ..., j_n\}.$$

Each job, j_i comprises a sequence of R_i tasks.

$$j_i = \langle t_{i1}, t_{i2}, ..., t_{iR_i} \rangle,$$

which have to be processed in order with a start date $s(j_i) \in \mathbf{Z}^+ \cup \{0\}$ and a due date $d(j_i) \in \mathbf{Z}^+$ determining the time when the job must be completed.

A task, t is an ordered pair $(m(t), l(t)) \in \mathrm{M} \times \mathbf{Z}^+$ where $m(t) \in \mathrm{M}$ is the machine on which it has to be performed and $l(t) \in \mathbf{Z}^+$ represents the time required to process it. Let T denote the set of all tasks

$$T = \{t_{11}, ..., t_{1R_1}, t_{21}, ..., t_{2R_2}, ...t_{n1}, ..., t_{nR_n}\}$$

A schedule is a function $S : T \rightarrow Z^+$ with constraints :

1. For all tasks, $t \in \mathbf{T}$, if $s(t) = s$ and $m(t) = m$ then $\nexists\, \acute{t} \in \mathbf{T}, s.t.$ $s(\acute{t}) \in \{s, s+1, \ldots, s+l(t)\}$ and $m(\acute{t}) = m$

(ie. we never have two tasks running at the same time on the same machine. Note,we are assuming once a task starts, it is not interrupted by another task. Such schedules are called non-preemptive).

2. For all jobs, $j \in J$, if $j = \langle t_1, t_2, \ldots, t_R \rangle$ then $s(t_i) \geq s(t_{i-1}) + l(t_{i-1})$ for $i = 2, \ldots, R$ and $s(t_1) \geq s(j)$. We say a schedule is satisfactory if it sastifies $\forall j_i \in \mathbf{J}, s(t_{iR_i}) + l(t_{iR_i}) \leq d(j_i)$.

3.3 Example

Example with three machine. (Please see Appendix A).

job 1 comprises : $\langle t_{11}, t_{12}, t_{13} \rangle$

job 2 comprises : $\langle t_{21}, t_{22}, t_{23}, t_{24} \rangle$

job 3 comprises : $\langle t_{31}, t_{32}, t_{33} \rangle$

In this case, all jobs have start date 0 and due date 8 and a satisfiying schedule has been found. A job shop schedule is optimal if $F = max(t_{iR_i}) + l(T_{iR_i}) : 1 \leq i \leq n$ is minimized over all schedules.

For any job i,length of the longest job i, $l(i) = \sum_{j=1}^{R} l(t_{ij}$ $F \geq max\{\lceil \frac{\sum l(t):t \in \mathbf{T}}{m} \rceil ,max \quad 1 \leq i \leq n \quad \sum l(t_{ij}) \quad | \quad 1 \leq j \leq R_i\}$

So that the above schedule in Appendix A is optimal since $F = \quad 8 = l(t_{21}) + l(t_{22}) + l(t_{23}) + l(t_{24})$

We say a schedule is tight if $\forall t \in \mathbf{T}, s(t)$ cannot be reduced without violating one of the constraints. This means that we will never idle on a machine and then start a task t if that task could be started earlier. Note that, if we adapt a greedy algorithm whereby a machine always processs an available task when it is idle, we can get poor schedule. For example, consider Gantt chart in Appendix B.

From the first Gantt chart in Appendix B, we get a bad schedule because the length of the schedule is 2R + 1 and the last task will only finish after due time while the scond Gantt chart shows, if we can idle at first then we can start the small task (the start time = 1) first rather than do the big task and we have delayed all other tasks. This schedule is a good schedule and the length of the schedule does not exceed the due time.

Note that the optimal schedule may not be satisfactionary. In such a case, we may seek a schedule which minimizes the number of late jobs, ie. jobs where final tasks do not complete until after their due time. This dissertation focuses on determining an optimal or near optimal schedule.

3.4 Scheduling Algorithms

Consider an algorithm whereby the tasks of a job J_i are scheduled in the given order, i.e., $\langle t_{11}, t_{12}, t_{13}, ..., t_{1k} \rangle$. The algorithm will never schedule t_{ij} before t_{iR} if $j > R$

At the same stage of the algorithm, J_i will be partially scheduled, i.e. tasks $t_i, ..., t_{iP}$ will have been scheduled, $t_{iP+1}, \ldots, t_{iR}$ will have not been scheduled. Thus, gives a way of practising the scheduling. if job_i has release time $r_i \in \mathbf{Z}^+$ and due time $d_i \in \mathbf{Z}^+$. Then if job_i comprises $\langle t_{11}, t_{12}, t_{13}, ..., t_{1k} \rangle t_{ij}$ must be done in the interval,

$$I_{ij} = \Big[r_i + l(t_{i1}) + l(t_{i2}) + ... + l(t_{ij-1}),\ d_i - \{l(t_{ij+1}) + l(tij + 2) + ... + l(t_{ik})\} \Big]$$

If $|I_{ij}| = l(t_{ij})$, then every task in job J must be scheduled as early as possible.

In order to generate a schedule, we must aware of the slackness of the schedule. We define the slack of t_{ij}, $sl_{ij} = |I_{ij}| - l_{ij}$. We want to prove that the slack for each task in the job is equal to the slack for the job. The slack for the job J_i is given by $sl(t_{ij}) = sl(J_i)$.

Proposition

If t_{ij} is a task in job Ji, then $sl(t_{ij}) = sl(J_i)$.

Proof

Assuming there are R jobs in J_i,

$sl(t_{ij}) = d_i - r_i - l(t_{i1} + ti2 + ... + tij)$

$= d_i - r_i - \sum_{k=1}^{R} l(t_{ij})$

$= sl(J_i)$

Example : Please refer to Appendix C

3.5 Instances

Instances of Job shop scheduling will be prepared in the following format[61]:

Each instance consists of a line of description. The first line will describe the number of jobs and the number of machines. Then each