5-2011

Shop problems in scheduling

James Andro-Vasko
University of Nevada, Las Vegas

SHOP PROBLEMS IN SCHEDULING

By

James Andro-Vasko

Bachelor of Science
University of Nevada, Las Vegas
2009

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Computer Science
School of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
May 2011

THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

James Andro-Vasko

entitled

Shop Problems in Scheduling

be accepted in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
School of Computer Science

Wolfgang Bein, Committee Chair

Lawrence L. Larmore, Committee Member

Laxmi Gewali, Committee Member

Ju-Yeon Jo Graduate Faculty Representative

Ronald Smith, Ph. D., Vice President for Research and Graduate Studies
and Dean of the Graduate College

May 2011

ABSTRACT

SHOP PROBLEMS IN SCHEDULING

By

James Andro-Vasko

Dr. Wolfgang Bein, Examination Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

The shop problems in scheduling will be discussed in this thesis. The ones I'll be discussing will be the flow shop, open shop, and job shop. The general idea of shop problems is that you're given a set of jobs and a set of machines. Each job is predeterminely broken into parts and there are rules to how each part is executed on a machine. In this thesis, several shop problems and their algorithms will be introduced that I have researched. There are several examples and counter examples that I have constructed. Also I will discuss how an arbitrary problem that can be solved polynomially can be changed so that there are no polynomial algorithms that can solve it. Scheduling is used in computer science in the area of operating systems and it can be used in engineering. This is an important for a company when they want to run several jobs efficiently so that resources can be saved.

TABLE OF CONTENTS

ABSTRACT iii

LIST OF FIGURES vi

CHAPTER 1 INTRODUCTION 1

CHAPTER 2 COMPUTATIONAL COMPLEXITY 2
- Classes P And NP 2
- Reductions 2

CHAPTER 3 FLOW SHOP PROBLEM 4
- Definition 4
- Minimizing Makespan 4
- Lemma 1 4
- Johnson’s Algorithm 5
- Example of Johnson’s Algorithm 7
- Degenerate Case with Johnson’s Algorithm 8
- Lemma 2 9
- Lemma 3 10
- Lemma 4 12

CHAPTER 4 OPEN SHOP PROBLEM 14
- Definition 14
- $O2||C_{max}$ 14
- Example 16
- Completeness of Algorithm 17
- $O|pmnt|C_{max}$ 20
- Network Flow Approach to Solve Preemptive Open Shop 23
- Example of Network Flow Algorithm 25
- Alternative Algorithm 26

CHAPTER 5 JOB SHOP PROBLEM 29
- Definition 29
- $J2|n_i \leq 2|C_{max}$ 29
- Example 31
- Alternative Algorithm 32
- $J|n = 2|C_{max}$ 33
- Theorem 37

CHAPTER 6 VARIOUS SHOP PROBLEMS 40
- Introduction 40
- Open Shop 40
- Flow Shop 41

CHAPTER 7 CONCLUSION..43

BIBLIOGRAPHY..44

VITA..45

LIST OF FIGURES

Figure 1 Visual for reductions 3
Figure 2 Example of schedule..5
Figure 3 Table of jobs 7
Figure 4 Schedule of jobs 8
Figure 5 Table of hypothetical jobs 8
Figure 6 Schedule of hypothetical jobs..9
Figure 7 Three possible cases if j is scheduled immediately after i...............................11
Figure 8 Case 1 schedule 15
Figure 9 Case 2 schedule 16
Figure 10 Schedule for $O2||C_{max}$ 17
Figure 11 Graph of all possible paths for open shop 17
Figure 12 Graph of one possible case for open shop 18
Figure 13 Graph of 7 by 7 matrix 22
Figure 14 Schedule of open shop jobs 22
Figure 15 Node property for network flow..24
Figure 16 Graph of network flow 25
Figure 17 Graph of the example 26
Figure 18 Schedule obtained from network flow 26
Figure 19 Bipartite graph..27
Figure 20 Schedule of counter example..28
Figure 21 Schedule for $J2|n_i \leq 2|C_{max}$ 32
Figure 22 Schedule with alternative algorithm..33
Figure 23 Optimal schedule for given jobs..33
Figure 24 Run of geometric algorithm 37
Figure 25 Graph of proof first case..38
Figure 26 Graph of proof second case 39

CHAPTER 1

INTRODUCTION

In this paper, various scheduling algorithms will be discussed. The scheduling problems that will be discussed heavily is the shop problems. Shop problems consist of jobs and each job is broken into several predetermined parts. The different shops have restrictions to how these parts can run. Before I go along, I should discuss the notation used in scheduling. The notation is $\alpha|\beta|\gamma$ where α represents the machine criteria. The ones discussed will be O, F, and J, which mean open shop, flow shop, and job shop respectively. A number following those letters describe an exact machine amount, otherwise there are arbitrary number of machines. β represents precedence constraints for example r_i which means there are release times and $pmnt$ which means preemption is allowed. γ means the optimality criteria which can be the sum of the completion time, $\sum C_i$ and the maximal completion time, C_{max}. There are many others as well, this represents a function to be minimized.

The algorithms, lemmas, and proofs are obtained through my research and I quoted them accordingly. My contributions in this paper were the examples and counter examples of the algorithms. I constructed a schedule the way the algorithm constructs them to have a better understanding of how the algorithm works and to see that the algorithms are efficient. I also discovered an alternative algorithm for these algorithms that would not yield an optimal result and showed the schedule constructed from that algorithm to see that it is not optimal. I also showed how some of the schedule as polynomial solvable and by modifying it slightly, the problem becomes NP-hard.

CHAPTER 2

COMPUTATIONAL COMPLEXITY

2.1 The Classes P and NP

One main issue in complex theory is to measure an algorithm's performance with respect to computational time. To measure the performance of an algorithm, we will need some form of an input. The input will be x which can be represented as $|x|$ which is the input length. Here are some encoding to the input length:

$$|x|_{bits} : \text{length of the binary encoding of } x$$

$$|x|_{max} : \text{magnitude of the largest number in } x$$

We measure the upper bound of x by $T(n)$. It is sometimes hard to calculate the exact value so we estimate by its asymptotic order. So we can say that $T(n) \in O(g(n))$ [Brucker, 2007]. Thus, instead of saying that the complexity is bounded by $2x^2+x-2$ is just simply $O(n^2)$ [Aho, 1974]. A problem is called **polynomially solvable** if there exists a polynomial p such that $T(|x|) \in O(p(|x|)$ for all inputs x for the problem.[Brucker, 2007] For example problem $J|n = 2|C_{max}$ is polynomially solvable. A problem is called **pseudo-polynomial** if $T(n)$ is polynomial where n is the input length with respect to the unary encoding [Aho, 1974]. It takes the form:

$$T(|x|) \in O(p(|x|_{bin}, |x|_{max}))$$

A problem is called **pseudo-polynomially solvable** if there exists a pseudo-polynomial algorithm that solves it. A **decision problem** is in which the output is {yes, no}. The class of all decision problems which are polynomially solvable is denoted as P. In decision problems, P can verify yes or no polynomially [Aho, 1974]. The NP can verify yes polynomially but not no.

2.2 Reductions

Decision problems can be used for reducing one problem to another. The form $P \rightarrow Q$ means that P reduces to Q. The way the decision problem can be used here is as:

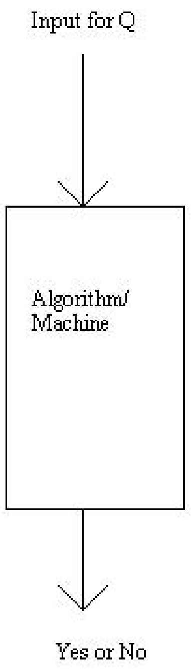


Figure 1: Visual for reductions

There is an input from P that goes into Q. Then Q attempts the validate it and return a yes or a no. This shows that Q has to be a harder problem than P. Since it is its own problem and it can solve another problem as well. A decision problem Q in NP is $NP - Complete$, if any $P \in NP$ then $P \rightarrow Q$ [Brucker, 2007].

CHAPTER 3

FLOW SHOP PROBLEM

3.1 Definition

The flow shop problem is a general shop problem in which:

- each job i consists of m operations O_{ij} with processing times p_{ij} $(j = 1, ..., m)$ where O_{ij} must be processed on machine M_j
- there are precedence constraints of the form $O_{ij} \rightarrow O_{i,j+1}$ $(i = 1, ..., m-1)$ for each $i = 1, ..., n)$. In other words, O_{i1} must be completed before O_{i2} can run and so on.

[Garey, 1976]

We want to find the job order π_j on machine j. We will discuss only the problems involving flow shop with the C_{max} objective function since it is the only objective function for flow shop the is not NP hard for arbitrary processing times. $\sum C_i$ is an example of an objective function which is NP hard for arbitrary processing times [Brucker, 2007].

3.2 Minimizing Makespan

The central idea of minimizing the makespan is to run several jobs on machines, efficiently, such that C_{max} is as small as possible. One of the only flow shop problems that are polynomially solvable is the $F2 \mid\mid C_{max}$ [Brucker, 2007]. The $F2$ means that this a flow shop and there are exactly two machines. Johnson's Algorithm can be used to find an efficient schedule to the $F2 \mid\mid C_{max}$ problem. The efficient schedule can be found if the sequence of jobs on both machines are identical, that is how Johnson's Algorithm finds the efficient schedule.

3.3 Lemma 1

For problem $Fm \mid\mid C_{max}$ an optimal schedule exists with the following properties:

1. The job sequence on the first two machines are the same.

2. The job sequence on the last two machines are the same.

[Brucker, 2007]

Proof:

Take an optimal schedule in which the processes order is the same on both machines for the first k jobs where $k < n$. Let the i-th job be the job following k. Then job j is a immediately successor of i on the first machine and not an immediate successor of i on the second machine. The following figure illustrates this situation.

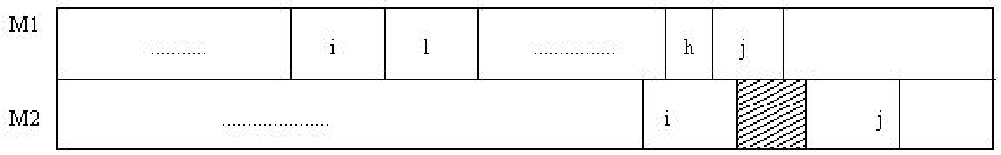


Figure 2: Example of schedule

If on machine 1 we shift job j to the position immediately after job i and move the rest of the jobs that follow by p_{j1} time units to the right, the C_{max} value will not change. Therefore the schedule will still be optimal. This contradicts the maximality of k. The second part of the lemma is proved similarly(just replace machine 1 with machine $n-1$ and machine 2 with machine n) [Garey, 1976].

3.4 Johnson's Algorithm

Now we will present Johnson's Algorithm for solving the 2 machine flow shop problem. The main part of Johnson's Algorithm is to find permutation or a list of jobs

$$L : L(1), ..., L(n)$$

such that this order is the same on machine 1 and machine 2, if the schedule follows this permutation then the makespan(C_{max}) is minimized [Brucker, 2007]. An optimal order is constructed by calculating a left list $T : L(1), ..., L(t)$ and a right list $R : L(t+1), ..., L(n)$, and then concatenating them to obtain $L = T \cdot R = L(1), ..., L(n)$. The lists T and R are constructed step by step.

At each step we find a job p_{ij} with the smallest processing time. If $j = 1$ then we put job at at the end of T, so we have $i \cdot T$. If $j = 2$ then we put job i at the beginning of R, so we have $i \cdot R$. We then remove job i from our set of jobs that haven't been processed yet. Then at the end we concatenate to form $L = T \cdot R$. Here is the formal sketch of the algorithm:

Johnson's Algorithm : $F2 \mid\mid C_{max}$

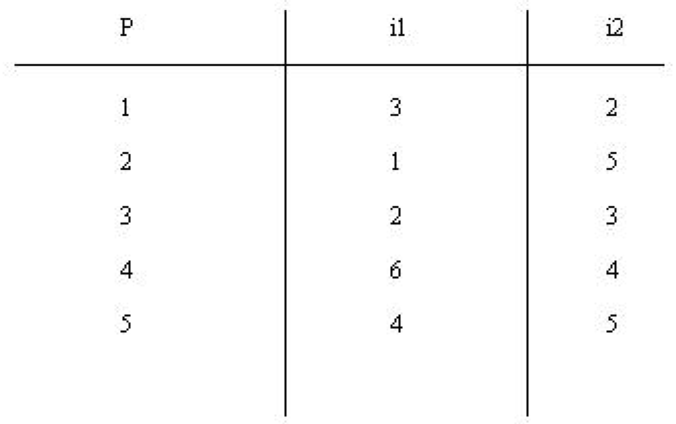
```
1. While $X = \{1, ..., n\}; T = \emptyset;\ R = \emptyset$
2. While $X \neq \emptyset$ DO
   BEGIN
3. Find $i^*, j^*$ with $p_{i^* j*} = \min\{p_{ij} | i \in X; j = 1, 2\}$;
4. If $j^* = 1$ THEN $T = T \cdot i^*$ ELSE $R = i^* \cdot R$;
```

```
5. $X = X - i^*$
   END;
6. $L = T \cdot R$
```

The * denotes i's or j's index not the actual processing time [Brucker, 2007].

3.5 Example of Johnson's Algorithm

Given jobs $p_{11} = 3, p_{12} = 2, p_{21} = 1, p_{22} = 5, p_{31} = 2, p_{32} = 3, p_{41} = 6, p_{42} = 4, p_{51} = 4, p_{52} = 5$. We can draw the following table to help construct the list.



P	i1	i2
1	3	2
2	1	5
3	2	3
4	6	4
5	4	5

Figure 3: Table of jobs

We can see that $T = \{2, 3, 5\}$ and $R = \{4, 1\}$ so the permutation $L = \{2, 3, 5, 4, 1\}$. Therefore, an optimal schedule has the same order on machine 1 and 2. The following schedule shows this.

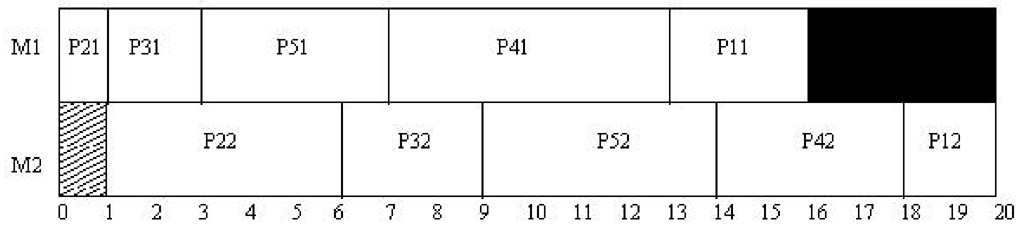
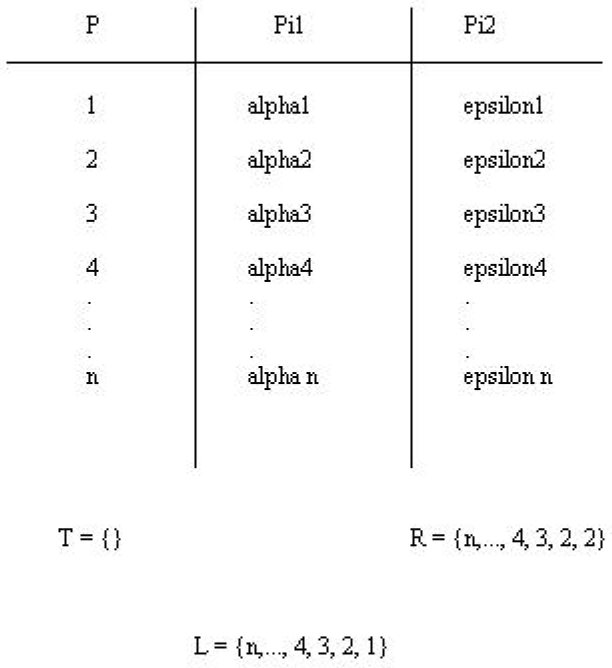


Figure 4: Schedule of jobs

3.6 Degenerate Case with Johnson's Algorithm

In this case our objective function is the sum of all the completion times. Johnson's Algorithm will always yield an optimum C_{max} value but doesn't necessarily work for $\sum C_i$. Here is an example of the degenerate case.



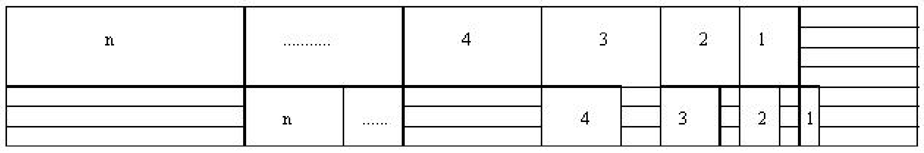
P	Pi1	Pi2
1	alpha1	epsilon1
2	alpha2	epsilon2
3	alpha3	epsilon3
4	alpha4	epsilon4
.	.	.
.	.	.
.	.	.
n	alpha n	epsilon n

T = {}

R = {n,..., 4, 3, 2, 2}

L = {n,..., 4, 3, 2, 1}

Figure 5: Table of hypothetical jobs



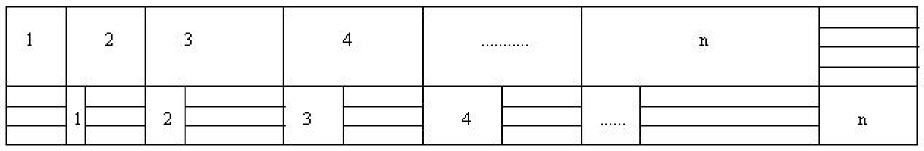


Figure 6: Schedule of hypothetical jobs

In this example the following inequality applies:

$$\epsilon_1 < \epsilon_2 < ... < \epsilon_n < \alpha_1 < \alpha_2 < ... < \alpha_n$$

Johnson's Algorithm will select all the ϵ jobs first and places them into R. Then $p_{n1} \rightarrow p_{n2}$ will run first and so on. But job n has the largest processing time which will hurt the $\sum C_i$ value. The efficient way to achieve the minimum $\sum C_i$ value is to run the job with the smallest processing time first. In this case reversing the order would be more optimal. The second schedule shows the optimal schedule for $\sum C_i$ value.

3.7 Lemma 2

Let $L = L(1), ..., L(n)$ be a list constructed by Johnson's Algorithm. Then

$$min\{p_{i1}, p_{j2}\} < min\{p_{j1}, p_{i2}\}$$

implies that job i appears before job j in L [Brucker, 2007].

Proof:

If $p_{i1} < min\{p_{j1}, p_{i2}\}$, then $p_{i1} < p_{i2}$ implies that job i belongs to T. If j is added to R, we have finished. Otherwise j appears after i in T because $p_{i1} < p_{j1}$. If

$p_{j2} < min\{p_{j1}, p_{i2}\}$, then $p_{j2} < p_{j1}$ implies that job j belongs to R. If i is added to T, we have finished implying that job i must appear before j in L. Otherwise j appears in R after i because $p_{j2} < p_{i2}$ implies that i appears before j in R, making i appear before j in L [Garey, 1976].

3.8 Lemma 3

Consider a schedule in which job j is scheduled immediately after job i. Then

$$min\{p_{j1}, p_{i2}\} \leq min\{p_{i1}, p_{j2}\}$$

implies that i and j can be swapped without increasing the C_{max} value [Brucker, 2007].

Proof:

If j is scheduled immediately after i, there are three possible cases:

1. p_{i2} starts during p_{j1} is running but doesn't finish before p_{j1}
2. p_{i2} starts at the same time as p_{j1} and p_{i2} finishes after p_{j1}
3. p_{i2} starts when p_{j1} is running and finishes after p_{j1}

The idea is that in all three cases, p_{i2} can only run once p_{i1} finishes. If the operations on machine 2, before p_{i2}, take longer than p_{i1} then machine 2 will not be idle. Otherwise machine 2 will be idle until p_{i1} finishes executing. Also if p_{i2} takes completes after p_{j1} then there will be no idle time on machine 2, otherwise machine 2 will be idle until p_{j1} finishes. The diagram describes the three cases.

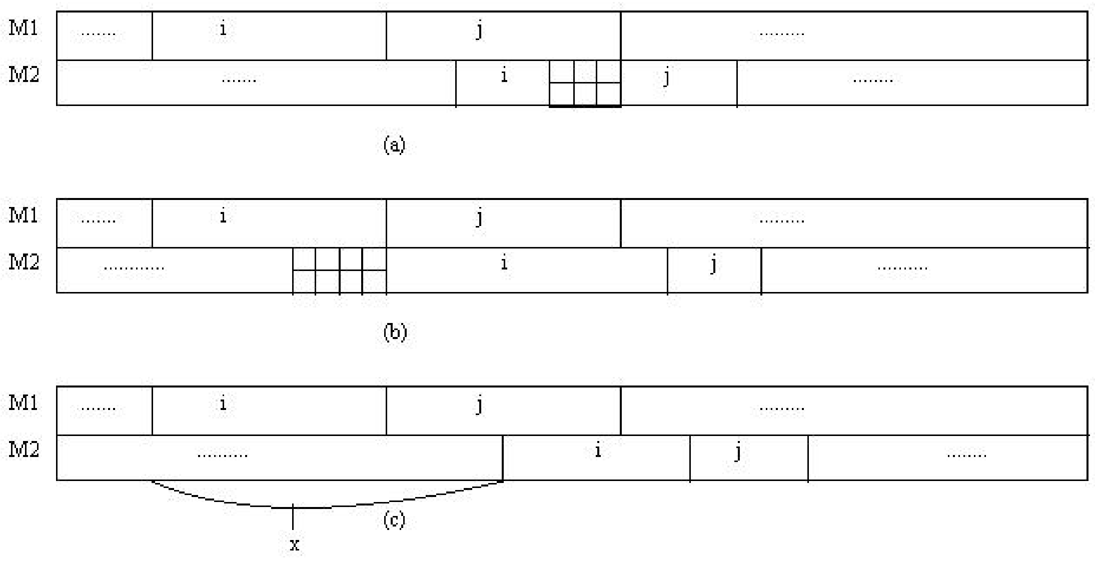


Figure 7: Three possible cases if j is scheduled immediately after i

Denoted by w_{ij} the length of the time period from the start of job i to the finishing time of job j in the situation. We have

$$w_{ij} = max\{\underline{p_{i1}} + p_{j1} + \underline{p_{j2}}, \underline{p_{i1}} + p_{i2} + \underline{p_{j2}}, x + p_{i2} + p_{j2}\}$$

The common terms can be combined under one *max* function.

$$w_{ij} = max\{p_{i1} + p_{j2} + max\{p_{j1}, p_{i2}\}, x + p_{i2} + p_{j2}\}$$

We have the following expression

$$w_{ji} = max\{p_{j1} + p_{i2} + max\{p_{i1}, p_{j2}\}, x + p_{i2} + p_{j2}\}$$

if i is scheduled immediately after j. The current lemma implies

$$max\{-p_{i1}, -p_{j2}\} \leq max\{-p_{j1}, -p_{i2}\}$$

We just multiply -1 to both sides of the inequality to get the above inequality. Adding $p_{i1} + p_{i2} + p_{j1} + p_{j2}$ to both sides of this inequality, we get

$$p_{i1} + p_{i2} + p_{j1} + p_{j2} + max\{-p_{i1}, -p_{j2}\} \leq p_{i1} + p_{i2} + p_{j1} + p_{j2} + max\{-p_{j1}, -p_{i2}\}$$

There is a property $a + b + max\{-a, -b\} = max\{a, b\}$, thus we get

$$p_{j1} + p_{i2} + max\{p_{i1}, p_{j2}\} \leq p_{i1} + p_{j2} + max\{p_{j1}, p_{i2}\}$$

which implies that $w_{ji} \leq w_{ij}$. Thus, swapping i and j will not increase the C_{max} value [Garey, 1976].

3.9 Lemma 4

Let the sequence $L : L(1), ..., L(n)$ constructed by Johnson's Algorithm is optimal [Brucker, 2007].

Proof:

Let Ψ be the set of all optimal sequences and assume that $L \notin \Psi$. Then we consider a sequence $R \in \Psi$ with

$$L(v) = R(v) \ for \ v = 1, ...s - 1 \ and \ i = L(s) \neq R(s) = j$$

where s is maximal. So the order of jobs in $L(n)$ are equal with $R(n)$ up to a certain point. Then job i is a successor of j in R since i doesn't appear in $R(s)$, therefore job i must appear after j in R. Let k be a job scheduled between job j and job i or $k = j$ in R. In L, job k is scheduled after job i. Thus, we must have

$$min\{p_{k1}, p_{i2}\} \geq min\{p_{i1}, p_{k2}\}$$

This holds for each such job k because k follows i so we can use the three cases to show that $w_{ik} \leq w_{ki}$. So by applying the last lemma to R, we may swap each immediate predecessor k of job i with i without increasing the objective value. We then get a sequence $R \in \Psi$ with $R(v) = L(v)$ for $v = 1, ..., s$ which contradicts the maximality of s. So the general idea behind this is that we have an optimal schedule, and a schedule that is equal to that optimal schedule to a certain point say s. Then, from the list that's not optimal, we can always swap any two jobs after s and the schedule will equal the optimal schedule after point s which is a contradiction [Garey, 1976].

CHAPTER 4

OPEN SHOP PROBLEM

4.1 Definition

An open shop problem is a special case of the general shop in which

- each job i consists of m operations O_{ij} $(j = 1, ..., m)$ where O_{ij} must be processed on machine M_j
- there are no precedence constraints between operation

[Brucker, 2007]

The problem is to find the job orders(operations belonging to a job) and the machine orders(orders to be processed on a machine)

4.2 O2||C$_{max}$

This is truly the only open shop problem, without preemption, which is polynomially solvable [Brucker, 2007]. The others are NP hard. The algorithm that solves this starts with two machines A and B. Processes that can run on A and B are a_i and b_i respectively with job i $(1, .., n)$ jobs. We define two sets I and J with the following properties:

$$I = \{i \mid a_i \leq b_i;\ i = 1, ..., n\}$$

$$J = \{i \mid b_i < a_i;\ i = 1, ..., n\}$$

Now we consider 2 cases that will give the optimal results.

Case 1:

$$a_r = max\{max\{a_i \mid i \in I\}, max\{b_i \mid i \in J\}\}$$

An optimal schedule is done the following way:

- all the $I-r$ jobs in an arbitrary order, jobs in set J in an arbitrary order, and then job r on machine A

- job r, jobs $I-r$ in the same order as on machine A, and then jobs in J with the same order as on machine A on machine B.

  [Gonzalez and Sahni, 1980]

The figure below illustrates this.

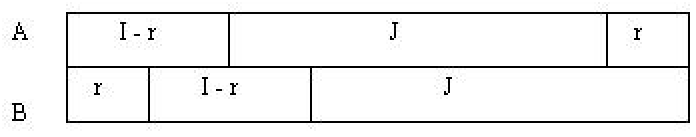


Figure 8: Case 1 schedule

Case 2:

$$b_r = max\{max\{a_i \mid i \in I\},\ max\{b_i \mid i \in J\}\}$$

An optimal schedule is done the following way:

- job r, jobs $J-r$ in an arbitrary order, and then I in an arbitrary order on machine A

- jobs $J-r$ in the same order as on machine A, jobs in I in the same order as on machine A, and then job r

[Gonzalez and Sahni, 1980]

The figure below illustrates this.

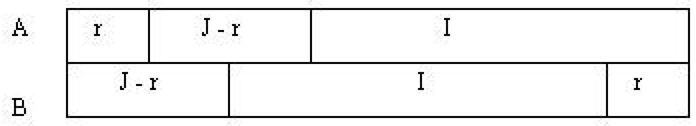


Figure 9: Case 2 schedule

4.3 Example

Here is an example of a schedule with the given jobs. The jobs are $p_{11} = 2, p_{12} = 1, p_{21} = 8, p_{22} = 5, p_{31} = 2, p_{32} = 4, p_{41} = 6, p_{42} = 3, p_{51} = 1, p_{52} = 3$. Now, all the a_i's are going to be the p_{i1}'s and the b_i's are going to be the p_{i2}'s. So we have the following list: $a_1 = 2, a_2 = 8, a_3 = 2, a_4 = 6, a_5 = 1$ and we have another list: $b_1 = 1, b_2 = 5, b_3 = 4, b_4 = 3, b_5 = 3$. Then based on the properties of sets I and J we have the following two sets

$$I = \{3, 5\} \; and \; J = \{1, 2, 4\}$$

We find a_r which is the maximum processing time from set I which is $a_3 = 2$. Then we find the b_r which is the maximum processing time from set J which is $b_2 = 5$. Out of a_3 and b_2, b_2 is the larger of the two so the r will be job 2. Since $b_r > a_r$ will apply the second case of our algorithm. When applied, the following schedule represents the optimal schedule.

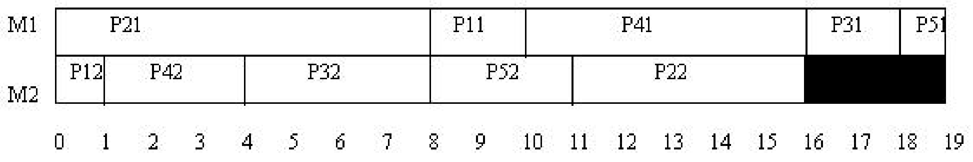


Figure 10: Schedule for O2||C$_{max}$

4.4 Completeness of Algorithm

Here we will show that this algorithm will always come up with the minimum makespan. We can look at the following graph to have a better visualization of how to pick a makespan.

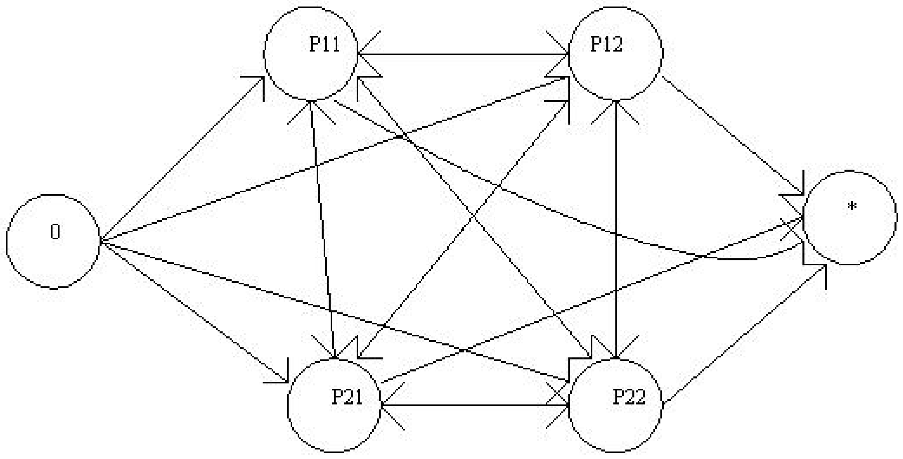


Figure 11: Graph of all possible paths for open shop

This graph shows all possible paths that be done for a typical open shop problem. A makespan of this is the path from 0 to all operations of one machine and the * [Brucker, 2007]. Here's an example of the makespan.

Here, a possible makespan is $0 \rightarrow p_{11} \rightarrow p_{21} \rightarrow *$. The order can be changed between $p_{11} \rightarrow p_{21}$ to $p_{21} \rightarrow p_{11}$. You can also have a possible path that involves machine 2. Now we need to apply to this to show the correctness of the $O2||C_{max}$.

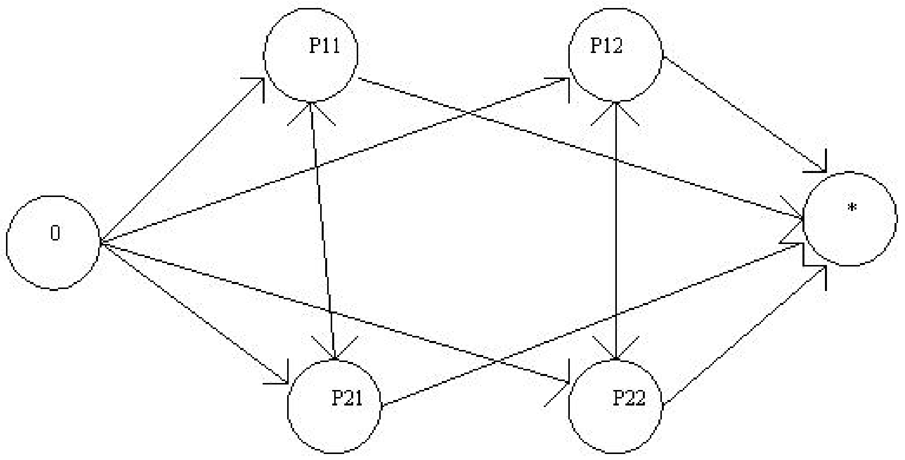


Figure 12: Graph of one possible case of open shop

We will look at each case separately and show that the makespan of the algorithm can never be worse than an actual makespan generated by the graph.

Case 1

There are two possible makespans, the larger of the two is the schedule's makespan. We take the path from the first machine and the second machine that the algorithm generates. They are

- $0 \to a_i(i \in I - r) \to a_i(i \in J) \to a_r \to *$
- $0 \to b_r \to b_i(i \in I - r) \to b_i(i \in J) \to *$

[Brucker, 2007]

The obvious path that any open shop problem can contain on machine 1 is $0 \to p_{11} \to p_{21} \to ... \to p_{i1} \to *$. For machine 2, the obvious path is $0 \to p_{12} \to p_{22} \to ... \to p_{i2} \to *$. The makespan for each of these is $\sum p_{i1}$ and $\sum p_{i2}$. Now we need to show that the makespan of the algorithm will never be greater than the makespan of the makespan that was just calculated. We set up the inequalities as such:

$$\sum_{i=1}^{|I|-1} a_i + \sum_{i=|I|}^{n-1} a_i + a_n \leq \sum_{x=1}^{n} p_{x1}$$

$$b_n + \sum_{i=1}^{|I|-1} b_i + \sum_{i=|I|}^{n-1} b_i \leq \sum_{x=1}^{n} p_{x2}$$

In the first inequality, we basically add up all the a_i's which is the first operations of all the jobs. So that's the same thing as if we just add up all the p_{i1}'s. In the second inequality, we basically add up all the b_i's which is the second operations of all the jobs which is the same thing as adding up all the p_{i2}'s. Unless there is a moment when either machine 1 or machine 2 has some idle times. Based on the way set A and set B are created, that will never happen. Therefore, in the first case, the schedule will always yield the minimal makespan.

Case 2

Like the previous case, there are two possible makespans, and the larger of the two is the lower bound C_{max} value. We take the path from the first machine and the second machine that the algorithm generates. They are

- $0 \rightarrow a_r \rightarrow a_i(i \in J - r) \rightarrow a_i(i \in J) \rightarrow *$
- $0 \rightarrow b_i(i \in J - r) \rightarrow b_i(i \in I) \rightarrow b_r \rightarrow *$

[Brucker, 2007]

Once again, the obvious path an open shop can have on machine 1 is $0 \rightarrow p_{11} \rightarrow p_{21} \rightarrow ... \rightarrow p_{i1} \rightarrow *$. For machine 2 the path is $0 \rightarrow p_{12} \rightarrow p_{22} \rightarrow ... \rightarrow p_{i2} \rightarrow *$.

The makespan for each of these is $\sum p_{i1}$ and $\sum p_{i2}$. Now we need to show that the makespan of the algorithm will never be worse than the makespan that was just calculated. The inequalities will look as

$$a_n + \sum_{i=1}^{|J|-1} a_i + \sum_{i=|J|}^{n-1} a_i \leq \sum_{x=1}^{n} p_{x1}$$

$$\sum_{i=1}^{|J|-1} b_i + \sum_{i=|J|}^{n-1} b_i + b_n \leq \sum_{x=1}^{n} p_{x2}$$

In the first inequality we add up the a_i's which is the all of the jobs' first operations. That is the same thing like adding up the p_{i1}'s . In the second inequality we just add up all the b_i's which is the same thing like adding up the p_{i2}'s. Just like in case 1, the machines will never be idle so trivially the inequalities will be equal, therefore this algorithm, in either case, will construct a schedule with a minimal makespan.

4.5 O|pmnt|C$_{max}$

This problem, due to the preemption, is polynomially solvable [Baptiste]. We first want to calculate the lower bound C_{max} value. If we create a schedule with a C_{max} value equal to the lower bound value, we have solved the problem. To find this value we have two values:

$$T_j = \sum_{i=1}^{n} p_{ij} \textit{ and } L_i = \sum_{j=1}^{m} p_{ij}$$

where n is the number of jobs, m is the number of machines, T_j is the total time needed on machine M_j, and L_i is the length of job J_i [Gonzalez, 1979]. The lower bound T is as follows:

$$T = max\{\max_{i=1}^{n} L_i, \max_{j=1}^{m} T_j\}$$

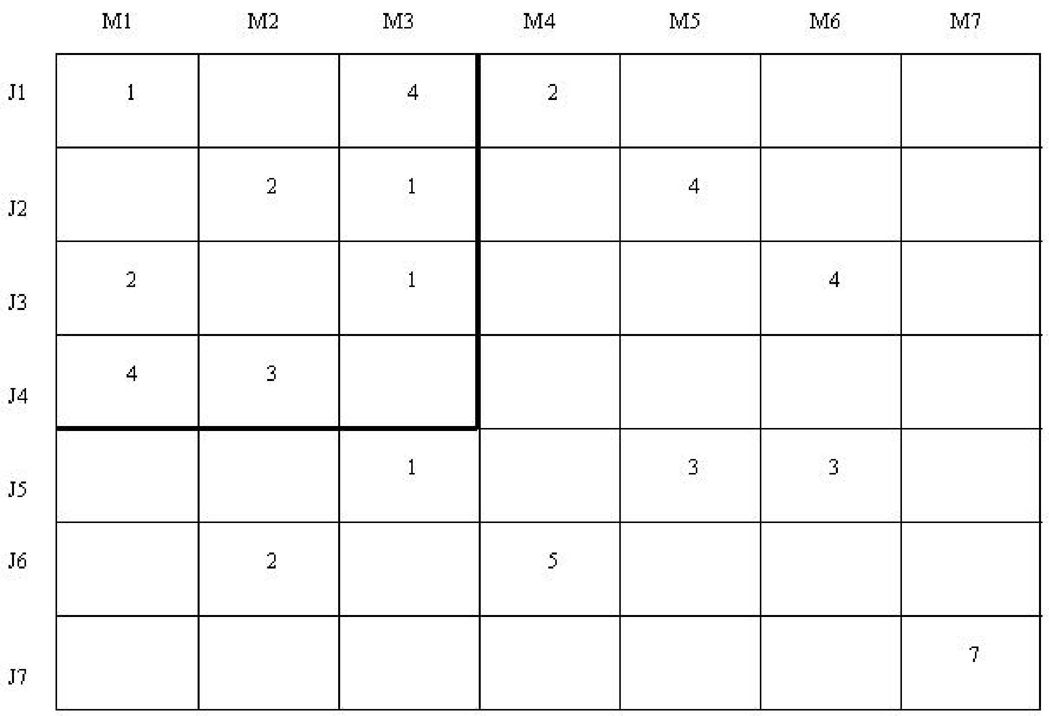
There are n jobs and m machines [Gonzalez, 1979]. We, then, make an $n + m$ by $n + m$ matrix. The columns of this matrix will be the jobs and the rows of this matrix will be the machines. There will be extra machines and extra jobs since it's larger than an n by m matrix.

To these extra rows and columns, we give those cells a value such that if each row and column is added, it will equal to T. Then we take a cell in each row in which neither of them have the same column index and put them into the appropriate machine. If a cell is chosen in which it's in an extra row or extra column, then do not place them into the schedule(otherwise put into schedule matching job with machine). After placed into the machine for one cycle subtract 1 to all cells, even those extra cells. Then this process is repeated until the n by m matrix has no more processing time remaining.

Example

In this example we have 4 jobs and 3 machines and processing times $p_{11} = 1,\ p_{13} = 4,\ p_{22} = 2,\ p_{23} = 1,\ p_{31} = 2,\ p_{33} = 1,\ p_{41} = 4,\ p_{42} = 3$ and all other $p_{nm} = 0$. We can construct a 7 by 7 matrix shown below.

We can calculate our T value to be 7. This will be the lower bound C_{max} value in this schedule. We can now select 7 jobs on 7 different machines. We can arbitrarily pick them. We can pick $J1$ on $M3$, $J2$ on $M2$, $J4$ on $M1$, and now we select the extra jobs or extra machines $J3$ on $M6$, $J5$ on $M5$, $J6$ on $M4$, and $J7$ on $M7$. This



	M1	M2	M3	M4	M5	M6	M7
J1	1		4	2			
J2		2	1		4		
J3	2		1			4	
J4	4	3					
J5			1		3	3	
J6		2		5			
J7							7

Figure 13: Graph of 7 by 7 matrix

will be the first cycle of our schedule. Any combination of extra job or machine does not get placed into a schedule. Every combination will be decremented by 1. The process goes until the highlighted box n by m matrix has no processing time left. Here is the final schedule below:

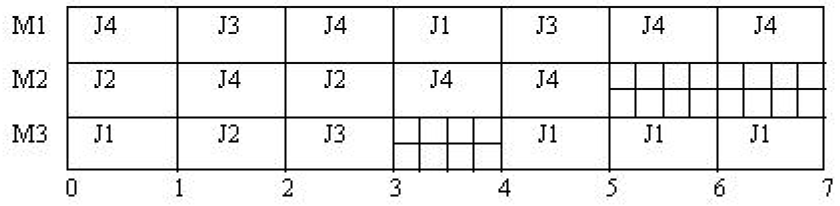


Figure 14: Schedule of open shop jobs

4.6 Network Flow Approach to Solve Preemptive Open Shop

The network flow can be used to solve this particular problem. It uses the same approach like the last algorithm. You need $n + m$ jobs (dummy jobs) and $m + n$ machines (dummy machines). Once again you have to calculate the L_i values and you have to calculate the T_j values. Then you have to find

$$T = \max\{\max_{i=1}^{n} L_i, \max_{j=1}^{m} T_j\}$$

Which is going to be the lower bound C_{max} value. Now we have to create a network N. The vertices of N will be

- a source s and a sink t
- job vertices J_i $(i = 1, ..., n + m)$
- machine vertices M_j $(j = 1, ..., n + m)$

[Ahuja, 1993]

The arcs are:

- for each J_i there is an arc (s, J_i) with capacity T and for each M_j and arc (M_j, t) with capacity t
- for each job J_i and each machine M_j with $p_{ij} > 0$, an arc (J_i, M_j) with capacity p_{ij}
- for each $i = 1, ..., n$ with $T - L_i > 0$ an arc (J_i, M_{m+i}) with capacity $T - L_i$ which connects jobs with a dummy machines
- for each $j = 1, ..., m$ with $T - T_j > 0$ an arc (J_{n+j}, M_j) with capacity $T - T_j$ which connects dummy jobs with machines

- for each dummy job and dummy machine an arc (J_{n+j}, M_{m+i}) with capacity to complete the network N

[Ahuja, 1993]

The last arc type is to complete the network such that the flow coming into a node must be equal to the flow coming out of the node. The following figure shows the property

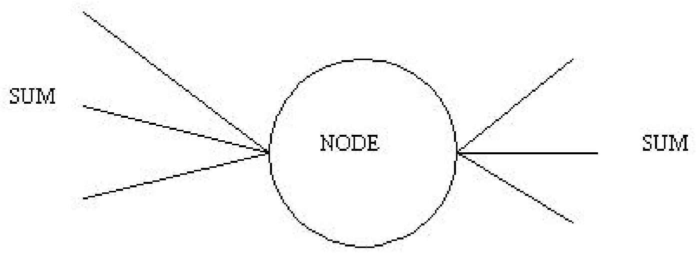


Figure 15: Node property for network flow

The following network will look something like the following figure.

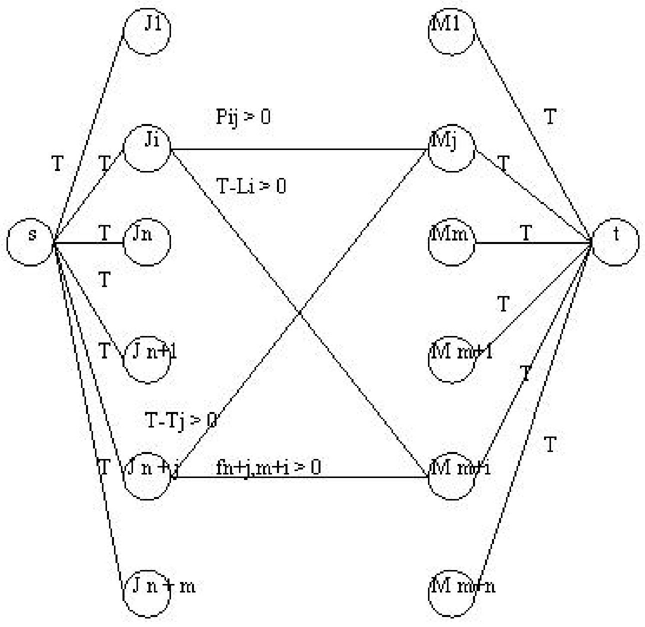


Figure 16: Graph of network flow

[Brucker, 2007]

Once you have the graph all worked out you have to apply the simple network flow mechanism. To draw the schedule, you pick one path from source s to a job J_i to a machine M_j to the sink t. For each arc from a job to a machine, you always want to pick the arc with the largest possible flow available. Then you find the minimum flow for all the arcs chosen as the preemption bound for a cycle. At each step, one arc will have no more available flow so in other words at each step an arc will be eliminated.

4.7 Example of Network Flow Algorithm

For this example, we will use the same values as from the previous example. The values are $p_{11} = 1, p_{13} = 4, p_{22} = 2, p_{23} = 1, p_{31} = 2, p_{33} = 1, p_{41} = 4, p_{42} = 3$. We can calculate the following values $L_1 = 5, l_2 = 3, L_3 = 3, L_4 = 7, T_1 = 7, T_2 = 5, T_3 = 6$. So it can be seen that $T = 7$, so that will be the lower bound C_{max} value. The following network can now be constructed:

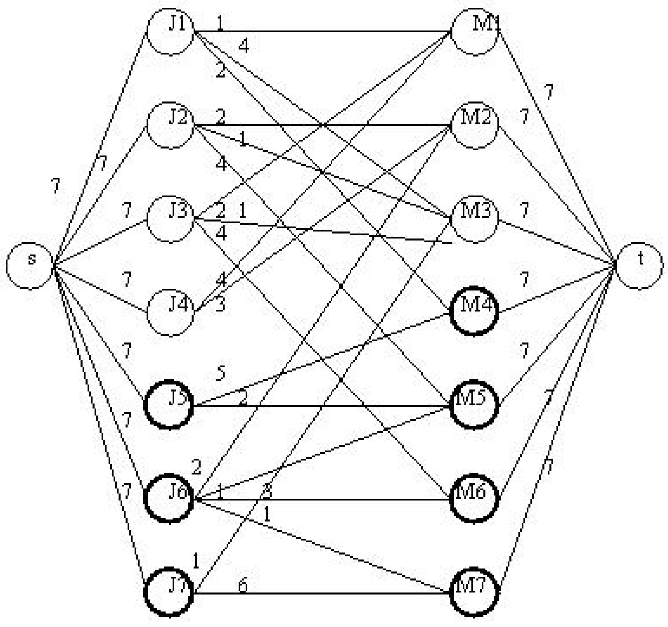


Figure 17: Graph of the example

The circles that have bold borders just denote dummy jobs or dummy machines. If we construct the schedule step by step we should have the following schedule constructed that will be obtained with the minimal C_{max}.

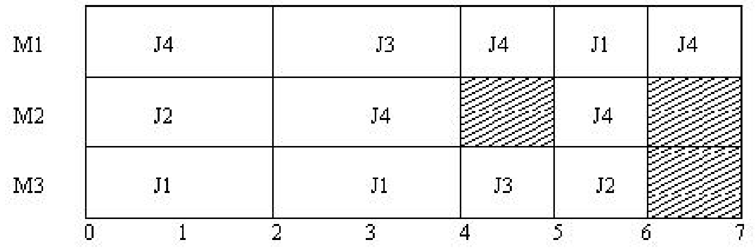


Figure 18: Schedule obtained from network flow

4.8 Alternative Algorithm

This starts with a bipartite graph. One side has all the jobs(i) and the other side has all the machines(j). Then a matching is done, if there are j machines then there are j arcs coming out of each job i and they will go to each machine j. A graph is shown below

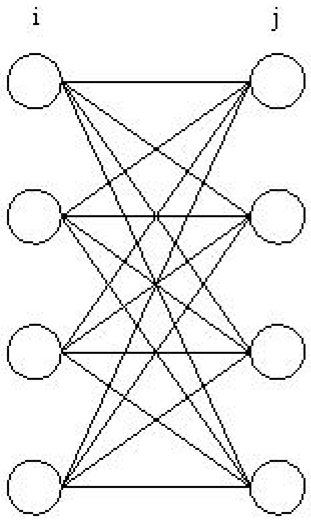


Figure 19: Bipartite Graph

The central idea behind this algorithm is picking the correct permutations. Initially for the first machine, use the graph to pick the first permutation. Then a table is created, each row represents a different machine and each column represents a job that runs on that machine. The row of permutations must have the following property:

$$\sum_{k=1}^{i} \text{ROW}_l\text{COLUMN}_k \cap \text{ROW}_2\text{COLUMN}_k ... \cap ...\text{ROW}_j\text{COLUMN}_k = \emptyset$$

In other words, if you take a column and check each row, we shouldn't have anything in common. Then we do this for all the columns. Now that we have the initial data, here is the algorithm:

1. start x = x mod i

2. for column x of our table place all the operations of all rows in column x

3. run those operations until one of them equals zero and subtract the run time from the rest of the operations

4. if all operations equal zero we are done else x = x + 1 go to step (1)

Here's an example of the algorithm. We have jobs $p_{11} = 3, p_{12} = 4, p_{13} = 2, p_{21} = 1, p_{22} = 5, p_{23} = 3, p_{31} = 2, p_{32} = 1, p_{33} = 4$ and have three machines. We can calculate the lower bound C_{max} value equals 10.

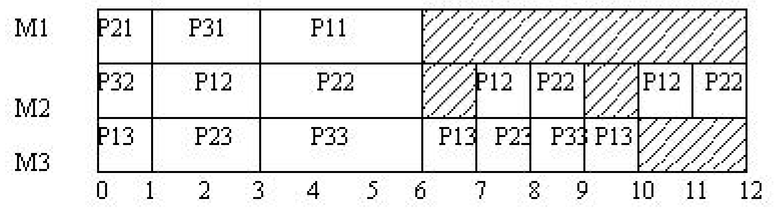


Figure 20: Schedule of counterexample

We can see that the C_{max} value = 12, therefore this algorithm will not always yield an optimal value for any arbitrary set of values.

CHAPTER 5

JOB SHOP PROBLEM

5.1 Definition

The job shop problem is a general shop problem with jobs $J_i (i = 1, ..., n)$ and machines $M_j (j = 1, ..., m)$ with the following properties:

- there are precedence constraints $O_{ij} \rightarrow O_{ij+1}$
- operation O_{ij} runs for P_{ij} time units on machine $\mu_{ij} \in M_1, ..., M_m$

[Brucker, 2007]

The idea is that operation 1 has to be completed before operation 2 can begin and there is no restriction to which machine operation j can run on.

5.2 J2$|n_i \leq 2|$C$_{max}$

This particular job shop problem with 2 machines and at most 2 operations per job can be used by reducing this two a 2 machine flow shop problem using Johnson's Algorithm. Before this reduction can be done, we must construct the following subsets first:

I_1: jobs which are processed only on machine 1

I_2: jobs which are processed only on machine 2

$I_{1,2}$: jobs which are processed first on machine 1 then on machine 2

$I_{2,1}$: jobs which are processed first on machine 2 then on machine 1

[Brucker, 1994]

Note that I_1 and I_2 contain jobs with only one operation. A remark to be made about this is that a job can start on machine 2 and then run machine 1. This is unlike the flow shop. On flow shop, O_{ij} must run on machine j the same goes for open shop. But on job shop O_{ij} must run on μ_{ij} where μ_{ij} could be on any available machine. In other words with job shop, O_{12} can run on machine 1, it doesn't have to run on machine 2. Now that we have constructed our subsets, we can use the following steps to to get an efficient schedule that will minimize the C_{max} value:

1. with the set $I_{1,2}$ construct an optimal sequence using Johnson's Algorithm and place that into $R_{1,2}$
2. with the set $I_{2,1}$ construct an optimal sequence using Johnson's Algorithm and place that into $R_{2,1}$
3. on machine 1 first schedule jobs $I_{1,2}$ according to order of $R_{1,2}$, then jobs I_1 in an arbitrary order, and then jobs $I_{2,1}$ according to the order in $R_{2,1}$
4. on machine 2 first schedule jobs in $I_{2,1}$ according to the order of $R_{2,1}$, then jobs in I_2 in any arbitrary order, and the jobs in $I_{1,2}$ according to the order of $R_{1,2}$

[Brucker, 1994] [Brucker and Kramer, 1996]

We can assume that in this schedule will always be active , i.e. there won't be a case in which both machines will be idle. If

$$\sum_{i \in I_{2,1}} p_{i2} \leq \sum_{i \in I_{1,2}} p_{i1} + \sum_{i \in I_1} p_{i1}$$

then there is no idle time on machine 1. Otherwise there is no idle time on machine 2. The reason is that if the sum of processing times for jobs on machine 1 in set I_1

and set $I_{1,2}$ will take longer than the jobs on machine 2 and machine 1 in set $I_{2,1}$ then the jobs in $I_{2,1}$ that run on machine 1 won't have to stop while it's p_{i1} first part runs because it is already finished [Brucker, 1994].

Now we will prove that the following schedule is optimal. If there is no idle time on machine 1 and no idle time on machine 2 or if

$$\max_{i=1}^{n} C_i = \sum_{i \in I_{1,2} \cup I_1} p_{i1} + \sum_{i \in I_{2,1}} p_{i2}$$

then we proved that it is optimal. Otherwise we restrict our problem to $I_{1,2}$ which is optimal using Johnson's Algorithm [Brucker, 1994][Peter Brucker and Sotskov, 1997].

5.3 Example

Now we will construct a schedule by using the previous algorithm. We are given the following jobs $p_{11} = 2, p_{12} = 3, p_2 = 3, p_3 = 4, p_{41} = 4, p_{42} = 3, p_5 = 5, p_6 = 7, p_{71} = 3, p_{72} = 1, p_{81} = 1, p_{82} = 6$. Now the sets will contain:

$$I_1 = \{P_2, p_3\}$$

$$I_2 = \{P_5, P_6\}$$

$$I_{1,2} = \{P_1, P_4\}$$

$$I_{2,1} = \{P_7, P_8\}$$

In set I_1 and I_2 the order can be arbitrary so the order will be the same in which it appears in I_1 and I_2. The order in $I_{1,2}$ and $I_{2,1}$ will be constructed using Johnson's Algorithm.

i/j	1	2
1	2	3
4	4	3

By using Johnson's Algorithm, we get the order $R_{1,2} = \{P_1, p_4\}$. Now we have the jobs in set $I_{2,1}$ can be represented by the following table.

i/j	1	2
7	3	1
8	1	6

By using Johnson's Algorithm on this, we can obtain the order $R_{2,1} = \{P_8, P_7\}$. Now we are ready to construct the schedule. If we just follow the algorithm, we will have the following schedule:

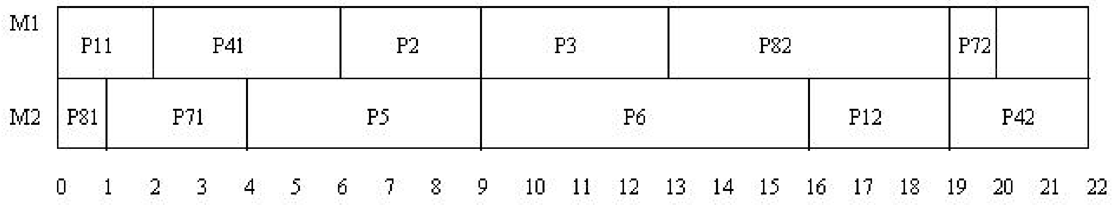


Figure 21: Schedule for $J2|n_i \leq 2|C_{max}$

5.4 Alternative Algorithm

Another way to construct a schedule for the $J2|n_i \leq 2|C_{max}$ problem can be done with a greedy−type algorithm as well. We can achieve this by following these steps:

1. create a two stacks s_1 and s_2 representing the order of jobs on machine 1 and 2 respectively.

2. take all the operations and sort them from smallest processing time to largest processing time and put into a queue Q.

3. starting from the beginning of Q pick the next available job and push onto s_1, the next iteration push onto s_2.

4. repeat step 3 until $Q = \{\emptyset\}$

This schedule would be optimal if the next job selected from Q is always from the front. This way there won't be any idle time on either machine. Otherwise it won't always yield the optimal C_{max} value. If $p_{i2} < p_{i1}$ then we can't select from the front of Q each time due to the job shop precedence constraints.

Counter example

Suppose we have jobs $p_{11} = 3, p_{12} = 5, p_2 = 1, p_{31} = 4, p_{32} = 2$. Then our queue $Q = \{p_2, p_{32}, p_{11}, p_{31}, p_{12}\}$. Then after we run the algorithm we would get $s_1 = \{p_2, p_{31}, p_{12}\}$ and $s_2 = \{p_{11}, p_{32}\}$. Here is the following schedule below.

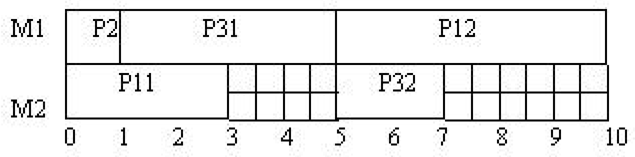


Figure 22: Schedule with alternative algorithm

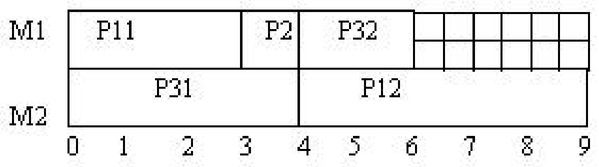


Figure 23: Optimal schedule for given jobs

The alternative algorithm will yield a C_{max} value of 10 and the optimal schedule has a C_{max} value of 9. So this shows that the alternative algorithm will not always give an optimal solution.

5.5 J|n = 2|C$_{max}$

This is a job shop problem with arbitrary number of machines and there are two jobs. We try to minimize the C_{max} value. This problem can be reduced to a shortest path problem. We first calculate the lower bound C_{max} value, if the shortest path equals the lower bound value, an efficient schedule can be constructed.

We put the processing times of job 1 on the x-axis and the processing times of job 2 on the y-axis. The intervals on both x and y axis are labeled by the machines on which they are to be processed. A feasible schedule corresponds to a path from 0 to F. Point 0 is just the origin of the graph and F is the pair a and b which is calculated:

$$a = \sum_{v=1}^{n_1} p_{1v} \text{ and } b = \sum_{v=1}^{n_2} p_{2v}$$

[Brucker, 1988]

The $\max\{a, b\}$ will give the lower bound C_{max} value. We want to find a shortest path from 0 to F with the following properties:

1. the path consists of horizontal, vertical, and diagonal lines(the diagonal lines are 45 degrees)

2. the path has to avoid the interior of any rectangle obstacles of the form I_1 x I_2 where I_1 and I_2 are intervals on the x-axis and y-axis which correspond to the same machine

3. the length of the path which is equal to the lower bound schedule length is equal to: length of horizontal parts + length of vertical parts + (length of diagonal parts) $/\sqrt{2}$

[Brucker, 1988]

We draw these lines such that none of these lines cross with the interior of any obstacles. To construct the shortest path we start from i(which is point O at the beginning) and draw a diagonal until we hit either the boundary of the rectangle formed by O and F or an obstacle. In the first case, if we hit the top of the rectangle, we go horizontally until we reach point F, or if we hit the far right of the rectangle then we go vertical until we hit point F. The second case is if the diagonal hits an obstacle then we have k which is a the SE corner of the obstacle and j is the NW corner of the obstacle. We than have two line segments (i, j) and (i, k). The length of (i, j) or $D(i, j)$ is equal to the horizontal or vertical piece plus the length of the projection of the diagonal piece on the x-axis or y-axis [Brucker, 1988].

In order to explain the algorithm we need to represent a few terms. We first order the forbidden regions according to lexicographic order of their NW corners. $D_i < D_j$ if for the NW corners (x_i, y_i) and (x_j, y_j) we have $y_i < y_j$ or $y_i = y_j, x_i < x_j$. We have r forbidden regions are indexed as:

$$D_1 < D_2 < ... < D_r$$

We have a set V which contains 0, F, and all the NW and SE corners of all forbidden obstacles. The shortest path will be denoted as d^* [Brucker, 1988]. Now here is the formal sketch of the algorithm.

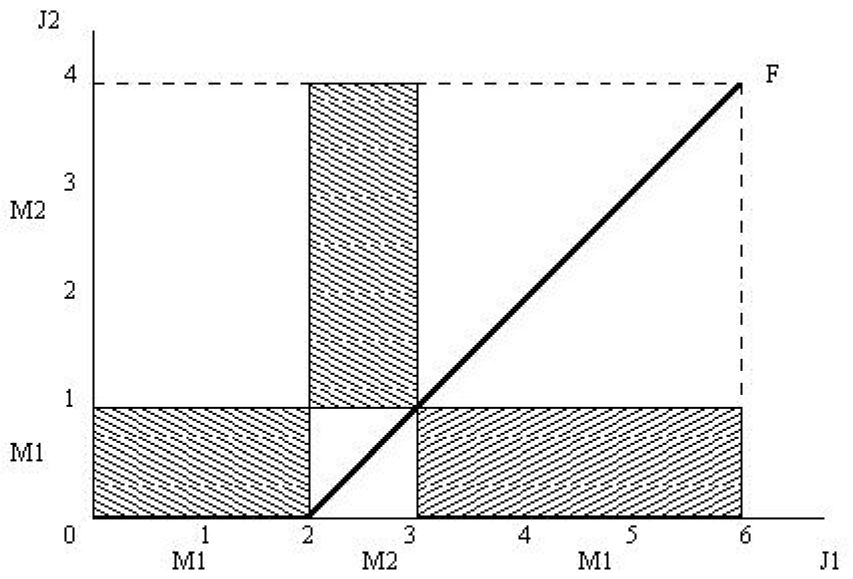
```
1. FOR ALL vertices i $\in V$ DO $d(i) = \infty$

2. FOR ALL successors $j$ of 0 DO $d(j) = d(O, j)$

3. FOR $i = 1$ TO $r$ DO BEGIN
```

```
4. FOR ALL successors $j$ of the NW corner $k$ of $D_i$

5. DO $d(j) = \min\{d(j), d(k) + d(k, j)\}$

6. FOR ALL successors $j$ of the SE corner $k$ of $D_i$

7. DO $d(j) = \min\{d(j), d(k) + d(k, j)\}$ END

8. $d^* = d(F)$
```

[Brucker, 1988]

Example of Algorithm

Given the following processes $p_{11} = 2, p_{12} = 1, p_{13} = 3, p_{21} = 1, p_{22} = 3$. We have the following solution



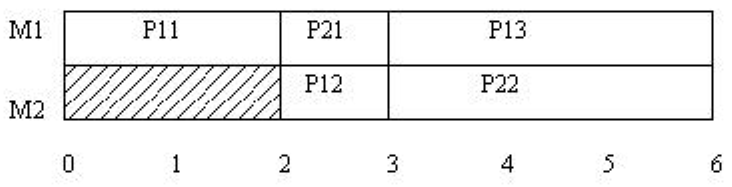


Figure 24: Run of geometric algorithm

$F = (6, 4)$, the lower bound C_{max} value is 6. It is trivial to see that the bold line's horizontal line and the projected diagonal over the x-axis is equal to the lower bound C_{max} value, therefore this graph represents an optimal schedule.

5.6 Theorem

A shortest path from O to F corresponds to an optimal solution of the shortest path problem with obstacles [Brucker, 2007].

Proof:

We know that the path from $O \rightarrow F$ is a path that avoids any obstacles, otherwise it would not be the shortest path. We consider the optimal solution p^* with longest starting sequence of arcs. If p^* is equals to this path, we have proved our case, otherwise assume that the last arc in this sequence ends at i. We would have the following situation:

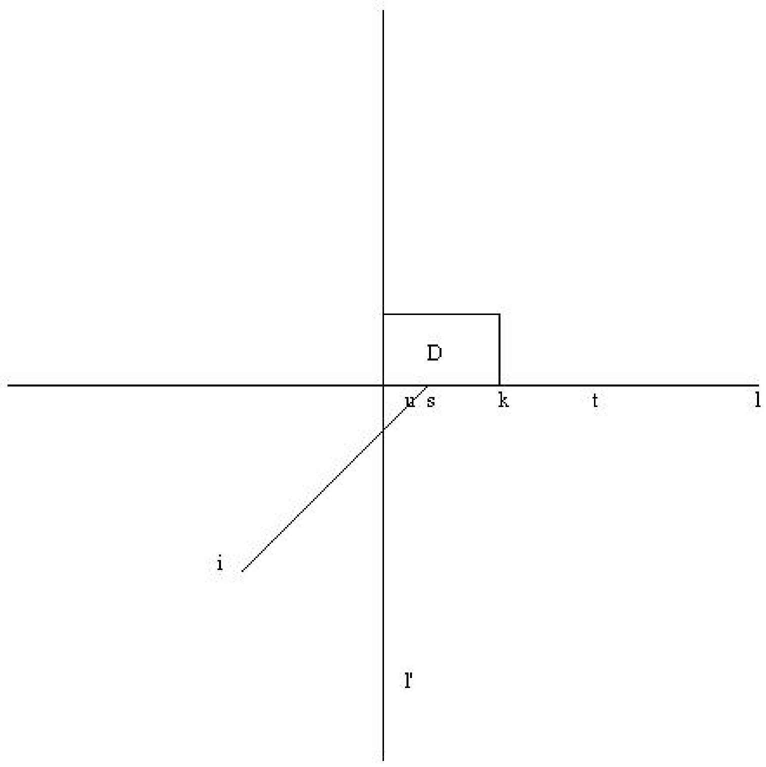


Figure 25: Graph of proof of first case

Let D be the obstacle we hit at some point s if we go in a NE direction from i. Line l is a line parallel to the x-axis and goes through point k which is the SE corner of D. Line l' is the line parallel to the y-axis and goes through point j which is on the NW corner of D. We denote the SW corner of D by u. Assume that s is on l and s' is on l'. Path p^* will cross line l at some point t. If $t = u, t = k$, or $t = k$ then we can replace the arc $i \to k$ by arc $i \to s \to k$ without increasing the length of p^*. If t is to the right of k then way replace arc $i \to t$ by arc $i \to s \to k \to t$ without increasing the length of p^* [Brucker, 2007].

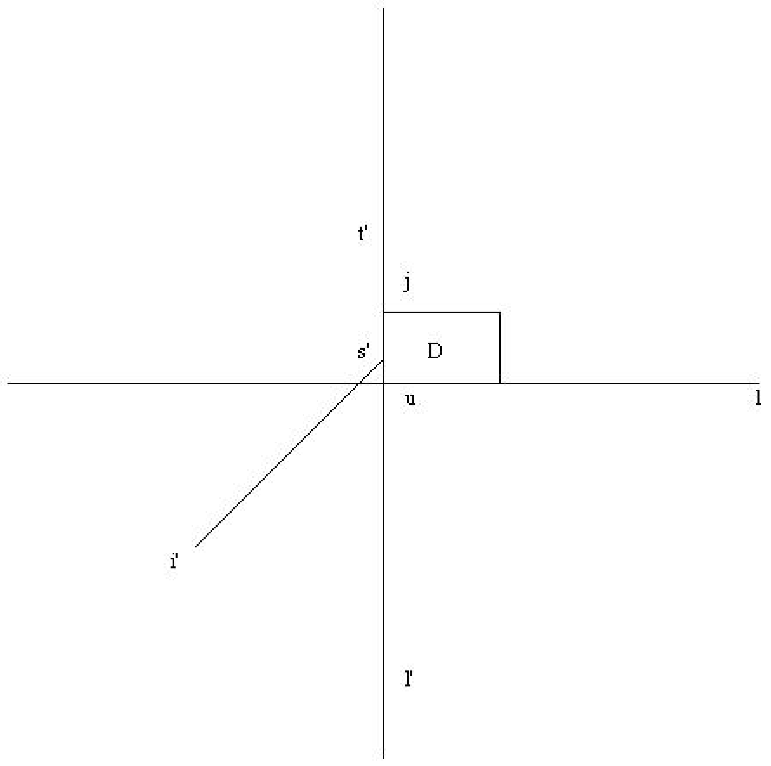


Figure 26: Graph of proof of second case

Now if the arc from i' crosses line l to the left of l and crosses l' at some point t' we have another set of cases. If $t' = u, t' = s'$, or $t' = j$ the we can replace $i' \to j$ by $i' \to s' \to j$ without increasing the length of p^*. If t' is above j then we can replace $i' \to t'$ by arc $i' \to s' \to j \to t'$ without increasing the length of p^*. Both of these cases contradicts the maximality assumption [Brucker, 2007].

CHAPTER 6

VARIOUS SHOP PROBLEMS

6.1 Introduction

In this section we will discuss that various scheduling problems that are polynomially solvable can become NP-hard just by modifying one of its constraints or objective functions.

6.2 Open Shop

We earlier showed that the $O2||C_{max}$ problem is polynomially solvable. If we were to change the objective to function and make the scheduling problem into $O2||L_{max}$, we would have an NP hard problem [Brucker, 2007]. To calculate L_{max} we have to calculate all the L_i's the following way

$$L_i = C_i - d_i$$

where d_i is the deadline for a particular job i. The reason it's NP hard is because we have to make the schedule such that we don't have any idle times on either machine 1 or machine 2. We also have to schedule the jobs such that we are not too late behind the deadline. If $O2||L_{max}$ is polynomially solvable then $O2||C_{max}$ must also be polynomially solvable. Lateness implies completeness. But this doesn't always work the other way around.

Like the previous example, $O2||C_{max}$ is polynomially solvable but if we add release times to this making it $O2|r_i|C_{max}$ is NP-hard [Brucker, 2007]. When we don't have release times, we can create the sets I and J and find the a_r and b_r and so on. But the problem with the release times is that we just can't schedule any job in any set because if job has a release time $r_i = 2$ and the algorithm schedules job i at time 1, it won't be allowed.

Another example is the $O|p_{ij} = 1|\sum U_i$ problem is polynomially solvable [Peter Brucker and Jurisch, 1993]. In this problem, you have U_i which is 0 if $C_i \leq d_i$ and 1 otherwise. So if the job is late to finish, there is a unit penalty, the idea is to try to finish the jobs before its deadline. If we were to add release times to this and have $O|p_{ij} = 1; r_i|\sum U_i$, now this problem is NP-hard [Kravchenko]. Once again we don't know how to construct the schedule. Release times are a factor but it can't influence the schedule order.

6.3 Flow Shop

We showed earlier that the $F2||C_{max}$ can be solved using Johnson's Algorithm. If we would add release times to the jobs, like in the open shop, we would get an NP-hard problem [Brucker, 2007]. Like in the open shop, if we schedule jobs in a certain way to minimize the C_{max} value the release times could get us into trouble. If we schedule a job i at time 2 but its release time $r_i = 5$, then we can't schedule the job at that time. We won't know that exact order of the jobs when release times are present.

Like in the open shop, if we take our polynomial problem $F2||C_{max}$ and replace the objective function with L_{max}, the problem $F2||L_{max}$ would be NP-hard [Brucker, 2007]. Same as in the open shop, we could construct an optimal schedule to minimize the C_{max} value but we have a set of due dates and they can be any arbitrary value. It would be hard to construct an algorithm that minimizes C_{max} and the L_{max}. You could construct all the possible schedules and one of them would be optimal but that wouldn't be found in polynomial time and hence it would be NP-hard.

We already know that $F2||C_{max}$ has a polynomial algorithm that maximizes the C_{max} value. If we would change the objective function to $\sum C_i$ which means the sum of all completion times, would be NP-hard [Peter Brucker and Sievers, 1994]. There is a branch and bound algorithm that finds **all** the possible permutations of the schedule and that would find the schedule that computes the minimal $\sum C_i$ value. This would obviously be an exponential algorithm algorithm making this NP-hard.

CHAPTER 7

CONCLUSION

In this paper, many different types of shop scheduling problems were discussed. But, there are many different algorithms that have no polynomial time algorithms that solve them. Only a few shop problems such as $O|pmnt|C_{max}$, $O2||C_{max}$, $J2|n = 2|C_{max}$, and etc are some of the ones that have been discussed. This seems like not many, but however, many of the shop problems, or most scheduling problems for that matter, are NP-complete. This is a uniquely strange area of computer science but, at the same time, a very interesting area in computer science.

Scheduling is an area studied by many different areas such as management, industrial engineering, and operations research. Good scheduling is an important part of the business world. If a scheduling algorithm is good, it can lower production and/or manufacturing cost which could keep a company competitive. This field started in the 1950s. Back then, many of these algorithms were very simple. This field has evolved a lot since then, now that many sophisticated algorithms have been developed.Scheduling plays an important role in implementing operating systems especially a long time ago when CPU, memory, and other resources were scarce [Leung, 2004]. Scheduling algorithms helped efficiently utilize these resources. Scheduling has come a long way but there is still much more to be discovered.

BIBLIOGRAPHY

A.V. Aho. *The Design And Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.

R.K. Ahuja. *Network Flows*. Prentice Hall, Englewood Cliffs, 1993.

P. Baptiste. Preemptive scheduling of identical machines. Technical Report 2000-314,Universite de Technologie de Compiegne, France, 2000.

Peter Brucker. An efficient algorithm for the job-shop problem with two jobs. *Computing*, 40:353-359, 1988.

Peter Brucker. A polynomial algorithm for the two machine job shop scheduling problem with a fixed number of jobs. *Operations Research Spektrum*, 16:5-7, 1994.

Peter Brucker. *Scheduling Algorithms*. Springer, Berlin, 2007.

Peter Brucker and A. Kramer. Shop scheduling problems with multiprocessor tasks on dedicated processors. *European Journal of Operational Research*, 90:212-226,1996.

M.R. Garey. The complexity of flowhsop and jobshop scheduling. *Mathematics of Operations Research*, 1:117-129, 1976.

T. Gonzalez. A note on open shop preemptive schedules. *Transactions on Computers,*28:782-786, 1979.

T. Gonzalez and S. Sahni. Open shop scheduling to minimize finishing time. *Journal of the Association for Computing Machinery*, 27:287-312, 1980.

Peter Brucker S.A. Kravchenko. Complexity of mean flow time scheduling problems with release dates. University Osnabruck.

Joseph Y-T. Leung. *Handbook of Scheduling*. CRC Press LLC, London, 2004.

B. Jurisch Peter Brucker and M. Jurisch. Open shop problems with unit time operations. *Methods and Models of Operations Research*, 37:59-73, 1993.

B. Jurisch Peter Brucker and B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49:107-127, 1994.

S.A. Kravchenko Peter Brucker and Y.N. Sotskov. On the complexity of two machine job-shop scheduling with regular objective functions. *Operations Research Spektrum*, 19:5{10, 1997.

VITA

Graduate College
University of Nevada, Las Vegas

James Andro-Vasko

Degrees:
Bachelor of Science, Computer Science, 2009
University of Nevada, Las Vegas

Thesis Title: Shop Problems in Scheduling

Thesis Examination Committee:
Chairperson, Wolfgang Bein, Ph.D.
Committee Member, Lawrence L. Larmore, Ph.D.
Committee Member, Laxmi P. Gewali, Ph.D.
Graduate Faculty Representative, Ju-Yeon Jo, Ph.D.