Minimizing Total Weighted Tardiness in a Two Staged Flexible Flow-shop with Batch Processing, Incompatible Job Families and Unequal Ready Times Using Time Window Decomposition

by

Anubha Alokkumar Tewari

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved July 2012 by the
Graduate Supervisory Committee:

John Fowler, Co-Chair
Lars Monch, Co-Chair
Esma Gel

ARIZONA STATE UNIVERSITY

July 2012

ABSTRACT

This research is motivated by a deterministic scheduling problem that is fairly common in manufacturing environments, where there are certain processes that call for a machine working on multiple jobs at the same time. An example of such an environment is wafer fabrication in the semiconductor industry where some stages can be modeled as batch processes. There has been significant work done in the past in the field of a single stage of parallel machines which process jobs in batches. The primary motivation behind this research is to extend the research done in this area to a two-stage flow-shop where jobs arrive with unequal ready times and belong to incompatible job families with the goal of minimizing total weighted tardiness.

As a first step to propose solutions, a mixed integer mathematical model is developed which tackles the problem at hand. The problem is NP-hard and thus the developed mathematical program can only solve problem instances of smaller sizes in a reasonable amount of time. The next step is to build heuristics which can provide feasible solutions in polynomial time for larger problem instances. The basic nature of the heuristics proposed is time window decomposition, where jobs within a moving time frame are considered for batching each time a machine becomes available on either stage. The Apparent Tardiness Cost (ATC) rule is used to build batches, and is modified to calculate ATC indices on a batch as well as a job level.

An improvisation to the above heuristic is proposed, where the heuristic is run iteratively, each time assigning start times of jobs on the second stage as due dates for the jobs on the first stage. The underlying logic behind the iterative approach is to improve the way due dates are estimated for the first stage based on assigned due dates for jobs in the second stage.

An important study carried out as part of this research is to analyze the bottleneck stage in terms of its location and how it affects the performance measure. Extensive experimentation is carried out to test how the quality of the solution varies when input parameters are varied between high and low values.

ACKNOWLEDGMENTS

My research, presented in the printed pages of this thesis signifies much more than the work I did in the duration of my graduate work at Arizona State University. It serves as a strong reminder of all that I was able to accomplish because of the amazing people that I have met and who have inspired me in many ways.

First and foremost I would like to extend my gratitude to Dr. John Fowler, my professor and co-chair of my thesis committee. He has been instrumental in encouraging me to pursue my interest in research in deterministic scheduling. I would like to thank him for his able guidance and continued support, which helped me complete my thesis.

I am very grateful to Dr. Lars Moench for his support and constant encouragement throughout the time that I have worked on my thesis. Dr. Moench was kind enough to serve as the co-chair on my thesis committee as well. He has been a great source of knowledge and help, always providing direction and valuable feedback at every juncture in my road to completion.

My thanks and appreciation goes to my committee member, Dr. Esma Gel for her time and support, not just in the course of my research but throughout my time at ASU.

Michael Clough and Siddharth Sampath, have served as more than colleagues, time and again providing me with very important insights and helping me understand a lot of complex concepts involved in my research.

I would like to take this opportunity to thank Raj Nooti and Michael Britman of US Airways, for giving me the opportunity to apply the knowledge I have gained during my masters and also for their compassionate understanding of my hectic schedule during the final stages of my research.

I must include a special mention of gratitude to my roommate Soumya Poduri and my good friend Jaiditya Namburi for being my family while at school, being really supportive of my work and cheering me on.

Last but definitely not the least I will remain forever grateful to my parents, without their constant support and unconditional love I would have never been able to realize my true potential. They never stopped believing in me and that has truly helped me plough through my research.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

The environment under consideration for the purpose of this research is a deterministic scheduling problem based on a two-stage flexible flow shop with a goal of minimizing total weighted tardiness. The problem is complicated by adding jobs that belong to incompatible families that arrive at different times.

Overall the problem is classified as a FF2|batch incompatible, $r_i|\sum w_i T_i$ in the $\alpha|\beta|\gamma$ notation of Graham et al. (1979) [17]. This is a common environment seen in many manufacturing and packaging processes in industry. Problems with batching of incompatible families that minimize total weighted tardiness have been proven to be NP-hard. Moreover, since the case of FF1|batch incompatible, $r_i|\sum w_i T_i$ problem has already been proved to be NP-hard by Moench et al. (2005) [9] and this research has added several layers of complexity, it becomes obvious that it too is NP-hard.

An example of this environment is semiconductor wafer fabrication where jobs, called lots, need to be batched on parallel machines on consecutive stages. However, lots of different families cannot be put together in one batch because of process restrictions.

Oxidation and diffusion process in semiconductor manufacturing are examples of incompatible batch processes. Moench et al. (2005) [9] mention in their paper that these processes generally take a longer amount of time as compared to the other steps in semiconductor fabrication. Improvements in terms of reducing tardiness in even small amounts significantly improve the manufacturing process. This makes the problem environment investigated in the research very interesting.

Chapter 2

PROBLEM DESCRIPTION

The focus of this research is to develop heuristics which will provide feasible schedules that seek to minimize total weighted tardiness in a two stage flow-shop with jobs that belong to incompatible job families and have unequal (but deterministic) ready times. In other words, jobs have to be scheduled to be processed in batches on both stages and each stage has a bank of identical parallel machines. Jobs arrive at the first stage at deterministic, but not necessarily equal, arrival (ready) times. The problem is simplified by assuming that all jobs belonging to the same family will have identical processing times and that all machines are capable of processing all families. Due to this assumption the problem at hand takes the form of a flexible flow-shop. It must be noted however, that while each machine is capable of processing all types of job, jobs of different families cannot be batched together on account of differences in processing requirements. After the first stage, the jobs are re-batched in preparation for the second stage. An important assumption made here is that there is unlimited buffer space between the two stages.

Overall the problem is designated as FF2|batch incompatible, $r_i|\sum w_i T_i$ in the $\alpha \mid \beta \mid \gamma$ notation of Graham et al. (1979) [17]. Here, w_i is the weight of job *i* and r_i is its ready time.

The performance measure that will be minimized is total weighted tardiness. Weighted tardiness of a job is calculated as the weight (importance/priority) of that job times its tardiness. Tardiness is calculated as the non-negative difference between the completion time (C_i) and the due date (d_i) for a job, i.e. max (0, $C_i - d_i$). This measure is summed over all jobs in the instance to obtain total weighted tardiness.

The figure below represents the problem described above that is the focus of this research.

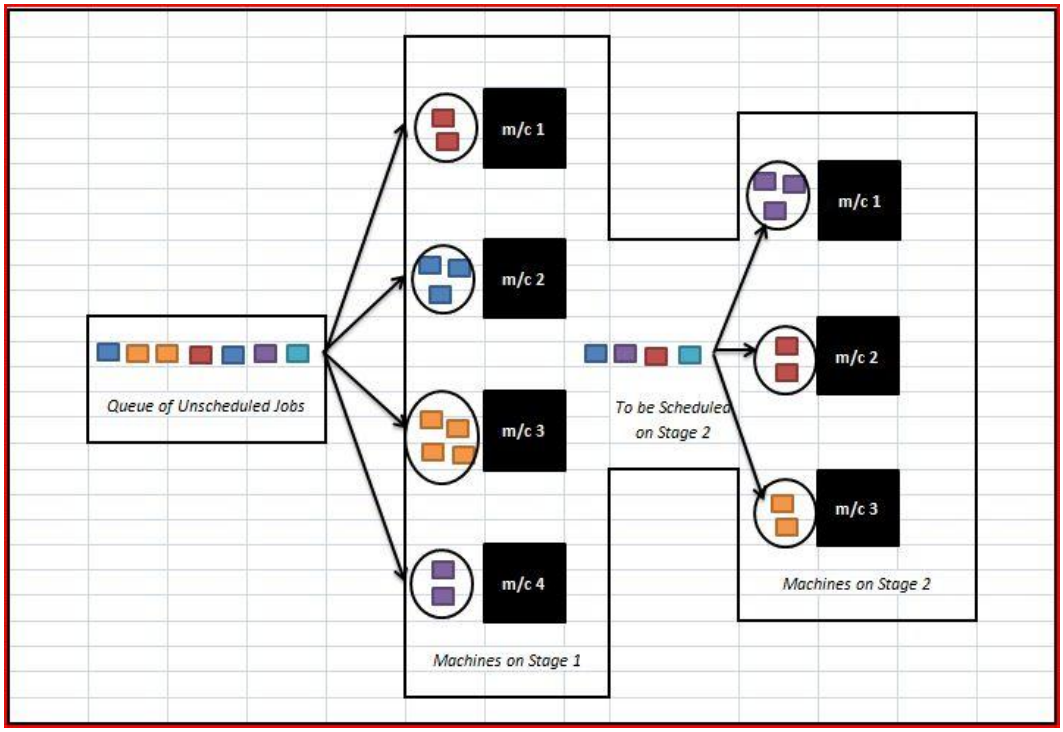

Figure 2-1 Schematic Representation of Problem

Since the job environment calls for unequal ready times, an important aspect of this research is to weigh the possibility of starting non-full batches against waiting to complete the batch depending on how the performance measure is affected.

In this research, we will test the heuristic for four different possible scenarios involving batching and serial processing (no batching) on parallel machines at each stage. The combinations tested will be serial-parallel batching, parallel batching-serial, serial-parallel batching and serial – serial (no batching on either stage). In the fourth case it becomes a two stage, parallel machine flow-shop which is still an NP-hard problem for the problem with total weighted tardiness as the performance measure.

Chapter 3

LITERATURE REVIEW

Within job scheduling, batching jobs in a flow-shop environment is a popular subject for research. A lot of work has been done in this field predominantly because of its implication and applied usefulness to the semiconductor fabrication processes.

Batching with incompatible families can be classified into either serial or parallel batching. Serial batching is where the total processing time of a batch is determined by the sum of all the processing times of the jobs that constitute the batch. On the other hand, in parallel batching the processing time of the batch is governed by the processing time required by the family to which the jobs of that batch belong i.e. all jobs belong to the same family and will have the same processing time which is also the processing time of the batch. Parallel batching is sometimes also referred to as "p-batching".

Cheng et al. (2004) [1] studied batching in a two stage flow-shop with dedicated machines in the second stage and minimizing make-span (F2|batching|Cmax) where F2 stands for a two stage flow-shop. Their work proposes an algorithm called CHECK which solves for a feasible solution by recursive computation of batch sizes and is solved in $O(n^F)$.

Kim et al. (2009) [2] consider the problem of a hybrid flow-shop with ready times and a product-mix ratio constraint ($F2|r_j$, compatible batching$|C_{max}$). Their paper suggests three algorithm, namely forward scheduling, backward scheduling and iterative search, and within each algorithm, different combinations of dispatching rules are used. The authors suggest that future scope for research includes development of local search or meta-heuristics to develop a solution that minimizes C_{max}.

C. J. Liao et al. (2008) [3] came up with a couple of mixed integer linear programs to tackle the problem of a two machine flow-shop with batching in two

scenarios one with waiting between the two stages and the other with no waiting or buffer between them. The paper considers p-batching. The performance measure they minimize is Cmax and the lower bound for each model is calculated by batching jobs by the LPT (longest processing time) rule on one machine and then taking the other machine into consideration by adding the processing time of the other machine to the cumulative completion time. Alternatively, they also propose a heuristic which uses a time limited version of the developed MILP models to compute near optimal solutions.

Cheng et al. (1996) [4] study an environment where “m” parallel machines exist per stage in a flexible flow-shop. The paper suggests a dynamic programming (DP) algorithm (initialized using shortest processing time). In the algorithm DP, batches containing consecutive jobs (arranged by SPT) are created and then scheduled on the machines from the end of the schedule to the front. The complexity of their algorithm is $O(mn^{m+1})$ where "n" independent and simultaneously available jobs are to be scheduled on “m” identical parallel machines. A lower bound is determined by simplifying the problem by assuming that the processing times are identical.

Oulamara (2007) [5] investigated a flow-shop environment with a parallel batching machine and job dependent set up times (F2|p-batch, no-wait|C_{max}). The paper proposes two algorithms to solve for optimal schedules using valued graphs based on processing times and weights assigned to jobs.

Hall et al. (2003) [6] studied a problem of the reverse nature where there is lot splitting in order to reduce the work in progress and lead times. The research done by the authors is interesting because of their approach to the problem. The paper suggests that the problem can be modeled into a Generalized Travelling Salesman problem (GTSP). Since the GTSP is not useful in larger problem instances the paper suggests a heuristic

algorithm in two phases that first uses a taboo search and then followed by the use of a greedy algorithm to minimize the makespan.

Brucker et al. (1998) [7] show that in parallel batching machines using some version of the SPT batch rule on each machine when there is no restriction on batch sizes is optimal to minimize makespan. However, with fixed batch sizes the dynamic programming algorithm used for a single machine can be generalized to give a pseudo-polynomial (in the sum of processing times) algorithm which can be used to solve for an optimal makespan. The authors also consider single batching machines where they give a forward dynamic programming algorithm with batch appending which is solved in $O(n^2P)$. This problem is binary NP-hard and considers p-batching.

Tang and Liu (2009) [8] considered a two stage flow-shop with one machine in each stage, where the first machine processes jobs one at a time and the second in batches. Jobs are assumed to have unequal ready times. The authors propose an MIP which solves for optimality in small instances and a combination of a Dynamic Programming algorithm for batching and heuristics based on dispatching rules to sequence batches.

This research is an extension of the research done by Moench et al. (2005) [9] where they study a single stage $P_m|$batch, incompatible$|\sum (w_iT_i)$ problem. They extend the research done by Balasubramanian et al. (2004) [10] and Moench et al. (2002) [11] in the area of solving for $P_m|$batch, incompatible$|\sum(w_iT_i)$ using genetic algorithms to include parallel machines in the flow-shop.

Moench et al. (2005) [9] develop two approaches to solve the $P_m|$batch, incompatible$|\sum (w_iT_i)$ problem. The first involves using the genetic algorithm (GA) to form batches and then schedule them on machines. In their second approach the authors propose using the GA to first assign jobs to machines and next forms batches and finally

sequences them using dispatching and scheduling approaches for the batching and sequencing portions. Additionally, they develop a time window decomposition heuristic which uses a modified version of the ATC rule. Only jobs which are ready in a time window of $t + \Delta t$ are considered to form batches and then scheduled on machines after the best combination of jobs that can form a batch is determined using ATC.

Klemmt et al. (2009) [12] considered a problem which deals with parallel batching machines in a single stage. They consider unequal ready times and unrelated parallel machines, where not all jobs can be processed on all machines. They compare the results of this mathematical model to solutions obtained by using variable neighborhood search and batching by ATC techniques. The MIP is used for all jobs within each time window and finds the optimal solution by considering all job combinations. Our research extends their mathematical model to incorporate a second stage of machines, identical parallel machines on each stage and incompatible job families where all batch sizes are the same.

In his doctoral dissertation Devpura [13] approaches the problem of solving for scheduling batches on parallel machines using a variety of techniques including dynamic programming, a decomposition heuristic, integer programming with column generation, ordering heuristics (like earliest due date, ATC, and weighted SPT) etc. However, the environment solved for does not include unequal ready times.

Chapter 4

PROBLEM ASSUMPTIONS

The assumptions of this research are similar to those made by Klemmt et al. (2009) and Moench et al. (2005). The assumptions made to simplify the problem at hand are:

1. There exist f job families. $F := \{1,\ldots, f\}$ represents the set of all families.
2. Jobs from different families cannot be processed as part of the same batch.
3. There are n jobs to schedule. $I := \{1,\ldots,n\}$ represents the set of all jobs.
4. There are m_1 and m_2 identical machines in parallel in stage 1 and 2, respectively. $M_1 := \{1,\ldots,m_1\}$ and $M_2 := \{1,\ldots,m_2\}$ represent the set of all machines for stages 1 and 2, respectively.
5. Machine preemption is not allowed.
6. The family of job i is represented as f_i.
7. The priority weight for job i is represented as w_i.
8. The due date of job i at the end of stage 2 is represented as d_i.
9. The processing time $p_{s,i}$ of job i on stage s is assumed to be equal on all machines in the stage. Also it is equal for all jobs of the same family for a single stage. Processing time is a function of family and stage.
10. The ready time of job i for stage 1 is represented as r_i. These may be unequal, but are deterministic i.e. known ahead of time.
11. The number of jobs that can be processed in one batch depends only on stage. This is defined by B_s, for stage s.
12. All families f can be processed on all machines on each stage.
13. The completion time of job i is denoted by C_i.

14. The weighted tardiness of job *i* is represented as $w_i T_i = w_i * max\ (C_i - d_i)^+$, where x^+ stands for the maximum of between x and 0.
15. Infinite buffer space exists between the two stages.

Chapter 5

MATHEMATICAL MODEL

The MILP formulated below is an extension of the MILP formulated by Klemmt et al. (2009) but has been modified and extended to fit the problem at hand. In the MILP proposed by Klemmt et al. mentioned above, the model is built to optimize total weighted tardiness on a single stage with "m" parallel machines. The MILP below solves our problem at hand - FF2|batch incompatible, $r_i|\sum w_i T_i$.

While extending the above model above to match this research, the model had to be modified to allow for all jobs to be processed on each machine. In the model indices *i, j, k, l* and *s* are used to depict job, batch, machine, family, and stage, respectively. Further, *J, K* and *L* stand for the maximum number of batches, machines and families in the problem instance. M (big M) is a very large number used to make the MILP linear in nature. The parameters used in the mathematical model are:

1. d_i = *Due date for job i on stage 2.*
2. w_i = *Weight of job i.*
3. $a_{l,i}$= *1 if job i belongs to family l, 0 otherwise.*
4. B_s= *Maximum batch size of stage s.*
5. $p_{s,i}$ = *Processing time for job i on stage s.*
6. r_i = *Ready time of job i to process in Stage 1.*

The decision variables used in the mathematical are:

1. $x_{i,j,k,s}$= *1 if job i belongs to batch j on machine k for stage s, 0 otherwise.*
2. $y_{j,k,l,s}$=*1 if batch j on machine k will process family type l for stage s, 0 otherwise.*
3. $s_{j,k,s}$ = *Start time of batch j on stage s on machine k.*
4. $C_{i,s}$ = *Completion time for job i in stage s.*

5. T_i = *Tardiness of job i.*

The MILP for FF2|batch incompatible, $r_i|\sum wiTi$ is as follows.

$$\text{Min } \sum_i w_i T_i \quad (5.1)$$

Subject to:

$$\sum_{k=1}^{K}\sum_{j=1}^{J} x_{ijks} = 1, \quad \text{For all i, s;} \quad (5.2)$$

$$\sum_{i=1}^{n} x_{ijks} \leq B_s, \quad \text{For all j,k,s;} \quad (5.3)$$

$$\sum_{l=1}^{L} y_{jkls} = 1, \quad \text{For all j,k,s, L=max for each job type;} \quad (5.4)$$

$$y_{jkls} - x_{ijks} \geq 0 \quad \text{For all i,j,k,s and l} \in \text{family of job i;} \quad (5.5)$$

$$x_{ijk1} * r_i \leq s_{jk1} \quad \text{For all i,j,k;} \quad (5.6)$$

$$s_{jks} + p_{si} * x_{ijks} \leq s_{jk+1s} \quad \text{For all i,j,}k_1,k_2\text{,s;} \quad (5.7)$$

$$M(1 - x_{ijks}) + C_{si} \geq s_{jks} + p_{si} \quad \text{For all i,j,k,s;} \quad (5.8)$$

$$C_{1i} \leq M(1 - x_{ijk2}) + s_{jk2} \quad \text{For all j,k;} \quad (5.9)$$

$$C_{2i} - T_i \leq d_i \quad \text{For all i.} \quad (5.10)$$

$$s_{jks}, C_{si}, T_i, x_{ijks}, y_{ijks}, \geq 0 \quad \text{For all i,j,k,s} \quad (5.11)$$

Equation (1) represents the objective function i.e. total weighted tardiness that this mathematical model is trying to minimize. Constraint set (2) ensures that each job is processed in exactly one batch and on exactly one machine on each stage. Maximum batch sizes on each stage are limited by constraint set (3). Constraint set (4) ensures that all jobs in a batch (on either stage) belong to the same family. Constraint set (5) ensures that a job is not part of a batch processing jobs from another family. Together constraint sets (4) and (5) ensure that only jobs of the same family are part of a batch. Constraints (6) and (9) make sure that a job is put on a machine in stage 2 only after it is ready or has

completed processing on stage 1. Constraint set (7) compels the model to start a new batch on a machine only after the previous one completes processing. Constraint set (8) limits the values that can be assigned to the completion time of each job. Constraint (10) determines the objective function (1) to be minimized. Constraint set (11) contains the non-negativity constraints.

Chapter 6

HEURISTICS DEVELOPED

The primary heuristic used in this research is a modification and extension of a batching and sequencing heuristic suggested by Moench et al. (2005) [9] for a single stage of parallel batching. This heuristic makes use of the ATC (apparent tardiness cost) index first proposed by Vepsalainen and Morton (1987) [14].

6.1 BATC Heuristic

The basic fundamental of this heuristic is to only consider jobs ready to be processed in a certain time window (t, t + Δt). The logic is that every time a machine becomes free on either stage, hypothetical batches are formed (one from each family) and one of these is chosen to be scheduled on the machine for processing. The important aspect remains that the batches mentioned above are only formed from the set of unscheduled jobs which are ready at a time less than the upper limit of the time window (t, t + Δt). The figure below depicts how a batch is chosen from all jobs ready in the time window t to t + Δt.

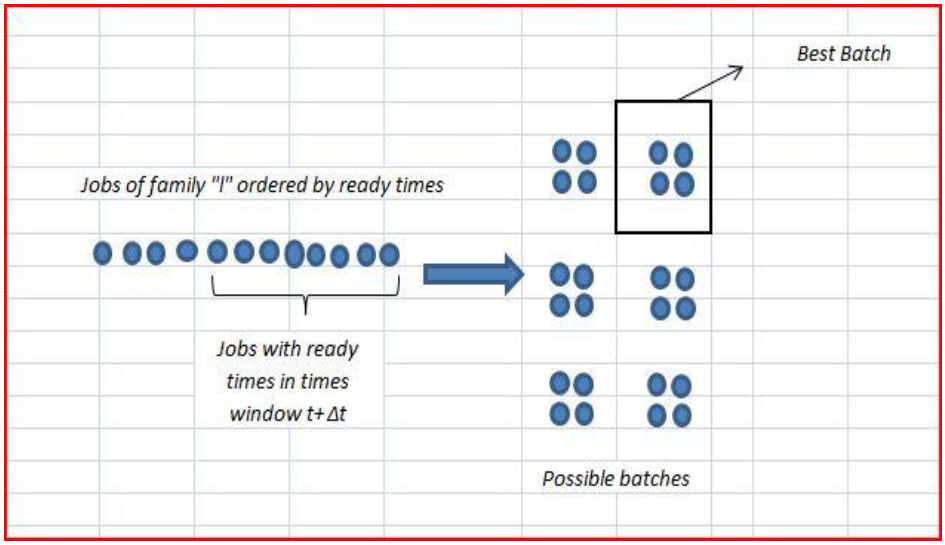

Figure 6-1 Implementation of Time Window

Using notation similar to that used by Moench et al. (2005), the set of unscheduled jobs from family l, with ready times that fall in the time window are designated by:

$$M(l, t, \Delta t) := \{il \mid r_{il} \leq t + \Delta t\} \quad (6.1)$$

Here, r_{il} stands for the ready times of job *i* of family *l*. Further, another set is defined based on the above set with an additional constraint which dictates the maximum number of jobs to be considered:

$$M`(l, t, \Delta t, thresh) := \{il | il \in M(l, t, \Delta t) \text{ and } pos(il) < thresh\}. \quad (6.2)$$

Pos(il) refers to the position of job i from family l with respect to I_{il}, where I_{il} is the criterion (ATC) for evaluating jobs. Thresh is a threshold used to calculate the maximum number of jobs i to be considered from family l within the time window (t, t + Δt) for batching purposes. In the case that the number of jobs in set *M`* exceeds the maximum batch size of the machine for which these batches are being formed, all possible batch combinations of family l, from set *M`* will have to be considered.

It is important to note that the amount of time taken to compute all possible batch combinations when a machine becomes available depends on the values we assign to Δt and thresh. The smaller the time window and thresh value, the fewer jobs will be considered for batching from a family l when a machine becomes available at time t.

The criterion used to evaluate batches is a modification of the ATC index described below, where $I_{il,ATC}$ is the index for job *i* of family *l* at time *t:*

$$I_{ik,ATC}(t) = \left(\frac{w_{ik}}{p_k}\right) \exp\left(\frac{-(d_{ik} - p_k + (r_{ik} - t)^+)}{k\bar{p}}\right) \quad (6.3)$$

Here, k in the denominator stands for the look-ahead parameter and $\bar{p}$ stands for the average processing time of the unscheduled jobs. Using the parameter thresh, *M`(l, t, Δt, thresh)* is formed from the set *M(l, t, Δt)* after it is ordered based on non-decreasing values of $I_{il,ATC}$.

From the above ordered set of jobs, *M'* we form possible combinations of batches for each family once a machine is available. A slightly modified version of the ATC rule called the BATC rule is used as a metric to evaluate best batches. This rule was suggested by Moench et al. (2005) [9] and is given below.

$$I_{bk}(t) = \sum_{i=1}^{n_{bk}} \left(\frac{w_{ik}}{p_k}\right) exp\left(\frac{-(d_{ik}-p_k-t+(r_{bk}-t)^+)^+}{k\bar{p}}\right) * \left(\frac{n_{bk}}{B}\right) \quad (6.4)$$

In the equation above, r_{bk} denotes the maximum ready time of jobs in the batch being formed, $r_{bk} = max_{i \in Bik}\ (r_{ik})$, n_{bk} is the number of jobs in the batch and B is the maximum batch size of the machine under consideration. The calculation of this index has two significant parts. One being, that, if a batch is not completely full the BATC index of the batch is reduced. This is done by appending the last term to the equation, which reduces the ATC index of the batch by multiplying it by the ratio of "fullness" (between 0 and 1) of the batch in the case that the batch is not completely full. Thus, by penalizing batches which aren't completely full this rules tries to increase the fullness of the batch. The other important point is the inclusion of the term $(r_{bk} - t)^+$ to the main slack term. This ensures that if the jobs which are to be included in a batch being formed are not ready, the importance/ATC index of the batch should be reduced/decreased.

The algorithm used to solve the two stage flexible flow-shop at hand using the two ATC indices/rules defined is given below:

1. First due dates are calculated for stage 1 for each job by calculating and assigning half the slack available to each stage. $Slack = max\ (0, d_{2i} - p_{1j} - p_{2j} - r_i)$ and $d_{1j} = (r_i + p_{1j} + Slack/2)$ where d_{1i} and d_{2i} are due dates for job i on stage 1 and 2, respectively.
2. A suitable value is chosen for Δt which is the time window within which unscheduled jobs are considered for batching. At a given time t, the set $M(l, t, \Delta t)$ is formed which, is then ordered by non-decreasing I_{il} index (calculated for all

unscheduled jobs of each family). From this ordered set, using an assigned value for the parameter thresh we form the set *M`(l, t, Δt, thresh).*

3. We then select the first machine *k*, that becomes available at a time less than or equal to time *t*. After this, BATC (batch ATC) values are computed for all possible batches (one of each family). From all the batches formed, the one with the highest BATC value is selected to be scheduled for processing on this machine *k*.

4. The new ready time for machine *k*, as well as completion times for jobs in the batch *j* that has just been processed are calculated. The new ready time for machine *k* becomes $max\ (t, r_{bl}) + p_{1j}$, where p_{1j} is the processing time of the family which forms batch *j* for stage 1. The above three steps are repeated until all jobs have been scheduled for processing on stage 1. Completion times for the jobs are calculated by adding processing time p_{1j} to current time of system/start time of batch.

5. Steps 1, 2, 3 and 4 are executed until all jobs are scheduled to be processed on stage 1.

6. Completion times on stage one for a job *i* become the ready times of the jobs for stage 2.

7. For stage two, sets *M* and *M`* are calculated at time *t*, given that values for *Δt* and *thresh* are taken to be the same as for stage 1.

8. Each time a machine *k* becomes available, from the set *M`*, possible combinations of batches are formed and the decision index BATC is calculated for each of them.

9. The batch with the highest BATC index is assigned to machine *k*. Using the processing time required for family *l* that batch *j* is from, we now calculate new ready time for machine *k* as well as new completion times for the jobs in batch *j*.

10. Repeat steps 6, 7, 8, and 9 iteratively to schedule all jobs for stage 2.

Considering the fact that the due date and slack of a job play a very important role in its batching process; we come up with an additional heuristic which, exploits this quality of the ATC and BATC indices. In the above heuristic due dates for stage 1 are calculated from given due dates of stage two simply by calculating slack values for each job and then assigning half the slack to each stage. Since this way of assigning due dates isn't as efficient in terms of distributing slack to the stage where it is needed more, the second heuristic aims to improve the method in which ATC and BATC are calculated by assigning more appropriate due dates for stage 1.

6.2 Iterative BATC Heuristic

The second heuristic is called the Iterative BATC approach. The important difference between this approach and the basic BATC heuristic described above is that each job undergoes two separate passes through the BATC heuristic. The first pass is as described in the heuristic above. However, before the second pass an essential adjustment is made to the calculated due dates of each job. From the first pass of the heuristic we have values for s_{2i} (the starting time of each job *i* on stage 2). Due dates for stage 1 are assigned these values, $d_{1i} = s_{2i}$. The fundamental logic behind this is that if a job doesn't need to start until a certain time *t* on stage 2 it doesn't have to complete processing on stage 1 until time *t* as well.

The main difference between the two approaches will be seen in those jobs which are important but have a greater amount of slack that can be assigned to stage 1 since there is a long waiting time between the two stages.

The algorithm used to implement the Iterative BATC approach is very similar to the BATC approach save a reassignment of values and is given below:

1. Steps 1 – 10 of the algorithm for the BATC heuristic are carried out as is.
2. New due dates are assigned for stage 1 for each job *i* as $d_{1i} = s_{2i}$, where s_{2i} is the starting time for job *i* on stage 2 as calculated by the BATC heuristic.
3. At a given time *t*, the set *M(l, t, Δt)* is formed which is then ordered by non-decreasing I_{il} index (calculated for all unscheduled jobs of each family). From this ordered set, using an assigned value for the parameter thresh we form the set *M`(l, t, Δt, thresh).*
4. We then select the first machine *k* that becomes available at a time less than or equal to time *t*. After this, BATC values are computed for all possible batches (one of each family). From all the batches formed, the one with the highest BATC value is selected to be scheduled for processing on this machine *k*.
5. The new ready time for machine *k*, as well as completion times for jobs in batch *j* that has just been processed is calculated. The new ready time for machine *k* becomes $max\ (t, r_{bl}) + p_{1j}$, where p_{1j} is the processing time of the family which forms batch *j* for stage 1. Completion times for the jobs are calculated by adding processing time p_{1j} to current time of system/start time of batch.
6. Steps 2, 3, 4 and 5 are executed till all jobs are scheduled to be processed on stage 1.
7. Completion times on stage one for job *i* become the ready time of the job for stage 2.
8. For stage two, sets *M* and *M`* are calculated at time *t*, given that values for *Δt* and *thresh* are taken to be the same as for stage 1.

9. Each time a machine *k* becomes available, from the set *M`*, possible combinations of batches are formed and the decision index BATC is calculated for each of them.

10. The batch with the highest BATC index is assigned to machine *k*. Using the processing time required for family *l* that batch *j* is from we now calculate new ready time for machine *k* as well as new completion times for jobs in batch *j*.

11. Repeat steps 7, 8, 9 and 10 iteratively to schedule all jobs for stage 2.

One drawback of running a single iteration of the Iterative BATC heuristic occurs because of the ATC index used to order jobs by priority. Since the new internal due dates are the starting times of stage 2, in the second pass of the Iterative BATC Heuristic, those jobs which are unimportant and are completed after their due dates may be even tardier in this pass.

This can be solved by running the Iterative BATC heuristic multiple times, (iteratively) until the value of the total weighted tardiness is not improved any further by running more iterations. The heuristic is coded such that the problem instance is passed through multiple iterations of the Iterative BATC until the value of the objective function does not change with further iterations. The iteration with the lowest value of total weighted tardiness is reported as the final solution for that problem instance.

Chapter 7

EXPERIMENTATION

The mathematical model (MILP) formulated as part of this research can provide a basis for the effectiveness of the BATC and Iterative BATC heuristics, but, only for relatively small problem instances. Being an NP-hard problem, if the problem instance to the MILP is increased beyond a certain size it becomes unsolvable in a reasonable amount of time using the mathematical model. The results are discussed in Chapter 8 i.e. Computational Results. This creates the need for another heuristic which can provide a reliable basis for comparison of solution quality when running larger problem instances.

7.1 Comparison Heuristics

Given the nature of the problem and the fact that the performance measure we are trying to optimize is total weighted tardiness, intuitively parameters like due dates, weights and release dates will have a greater impact on the order in which jobs should be processed. It is important to note that these heuristics are implemented only to provide some comparison to prove the effectiveness of the two BATC heuristics built. The four comparison heuristics that are used to test results assign jobs to machines based on due dates, release dates, ATC values and finally in random order. A detailed description of each one and its logic follows.

7.1.1 Batching and scheduling based on Earliest Due Dates (EDD):

Since changes in due dates directly affect the magnitude of tardiness of each job, an approach that processes jobs with earlier due dates will have a greater probability of reducing the overall tardiness. The way this approach works can be regarded as a dispatching rule with a non-greedy nature. As this approach does not use a time window approach there is no in-built decision metric which would allow a greedy implementation of this heuristic. The steps it follows are:

1. All jobs are sequenced in non-decreasing order of due dates for each job i, d_i. The due dates are due dates for the job at the end of the second stage.
2. At time t, when a machine becomes free the next job, based on due date priority in the sorted list is used to start a new batch. The next job with the same family l of job i, from the list is assigned to the newly started batch. This process is carried on until the batch reaches capacity. If enough jobs aren't available the batch is sent to the machine at less than capacity.
3. Finally, total weighted tardiness is calculated and summed across all jobs.
4. Steps 1 to 3 are repeated until all jobs are scheduled.

7.1.2 Batching and scheduling based on Release Dates (R_i):

This heuristic is based on FIFO (first in first out). It can be argued that on a dispatching heuristic level there is some value to processing jobs as they arrive without trying to apply foresight on how processing the current job affects the total weighted tardiness in the system. The underlying logic behind this approach is to process jobs in order of the arrival or ready-times r_i. It must be noted again that this dispatching heuristic does not use a time window approach to implement a greedy approach. The steps it follows are:

1. All jobs are sequenced in order of non-decreasing order of release dates for each job i, r_i. These are release dates for the jobs at the end of the second stage.
2. Repeat steps 2 – 4 from section 7.1.1 until all jobs are scheduled.

7.1.3 Batching and scheduling based on ATC (ATC_i):

Given that the parameter being optimized in our research is weighted tardiness, a very important comparison heuristic could be based on ATC values for jobs. ATC indices assigned to jobs give importance to both weight and the amount of slack available for each job. This approach works like a dispatching rule with a

non-greedy nature. As this approach does not use a time window approach there is no in-built decision metric which would allow a greedy implementation of this heuristic. The steps it follows are:

1. All jobs are sequenced in non-increasing order of ATC values for each job *i*, ATC_i. The ATC values are calculated based on a stage-wise basis given current time *t*, due date and ready times for the stage.
2. Repeat steps 2 – 4 from section 7.1.1 until all jobs are scheduled.

7.7.4 Batching and scheduling based on Random Order: The last comparison heuristic to be considered for sensitivity analysis is based on jobs being batched based on families but from a randomized sequence. It isn't expected that this heuristic will provide the smallest total weighted tardiness value, however, it could be useful to construct an upper bound on the value of the optimized solution. This heuristic is implemented by following these steps:

1. Jobs are sequenced in a randomized order, by assigning each job a random value and then sorting them based on these random values. Doing this will result in a random sequence of jobs which, does not take into account any other values of the job's attributes.
2. Repeat steps 2 – 4 from section 7.1.1 until all jobs are scheduled.

7.2 Test data generation and design of experiments

For the purpose of our research we use a method derived by combining the approaches of Moench et al. (2005) [9] and Yang et al. (2000) [15] for generation of problem instances. The logic behind building a hybrid method is to use the approach by Moench et al. where they build test sets for a single stage problem with parallel machines

and extend it to a second stage by incorporating ideas used by Yang et al. for testing heuristics in flexible flow-shops with multiple stages.

Montgomery [16] talks about the advantage of using 2^k factorial designs to test experiments in his book. Those problem instances are tested which use high and low values for each variable input parameter which might have a significant effect on the result of the experiment. In an attempt to use the significant advantage offered by 2^k designs one high and one low value is used for most parameters that are deemed to be important and would change the resultant outcome of the heuristic.

For the purpose of this research, we combine and select relevant input parameters used by Yang et al. and Moench et al, whose values will be varied between high and low. The parameters finally chosen are number of machines on stage 1 and 2, number of jobs/family, batch sizes on each stage, ready times, number of families, and due dates.

The cases of 3 and 5 parallel machines on both stages are used to generate problem instances. Weights are chosen for each job i, w_i from a uniform distribution over (0, 1). Next batch sizes for machines can be either 4 or 8 and 3 family types are considered for testing. We assume that the number of jobs/family can take values of either 10 or 15.

In their paper, Moench et al. use a parameter α, to define ready times r_i for jobs which uses an estimate of the makespan for each job based on their processing times, number of machines, batch sizes and an average batch utilization. Using similar logic, ready times are generated for jobs using the formula:

$$r_i \sim Uniform\ (0, \alpha * \frac{\sum_{i=1}^{n}(p_{1,i}+p_{2,i})}{mBf_{av}}) \qquad (7.1)$$

In the above formula, m is the total number of machines, B is the maximum batch size and f_{av} is the average batch utilization. Average batch utilization is set to 0.75 and α

is allowed to take values either 0.25 or 0.75. Using the calculated release dates (r_i), due dates for jobs at the end of stage 2 are calculated using the formula:

$$d_{2,i} = r_i + g_1 * (p_{1,i} + p_{2,i}) \quad (7.2)$$

In the above formula g_1, is called the tightness factor and can take values of either 1.1 or 1.5. Lower and higher values are chosen to be assigned to test tight and loose due dates, respectively. For each problem instance, processing times are randomly assigned to each of the three incompatible job families being considered by using a probability distribution shown in table 7-1. Since this research deals with a flexible flow-shop with two stages, there is some value in testing the efficiency of the heuristics built in situations where there is a bottleneck stage. Following the approach of Yang et al. (2000) [15], the first step is to calculate the average workload on machines for each stage using the processing times assigned above. Average workload is defined as:

$$W_{st} = \frac{\sum_{i=1}^{n} p_{st,i}}{m_{st} * B_s}. \quad (7.3)$$

In the above equation, *st* is the stage, m_{st} is the number of machines on this stage and B_{st} is the maximum batch size at the stage. The minimum workload across all stages is now called W_{min} and is taken as the basic (non bottleneck) stage. Conversely, the other stage where the value of the average workload per machine is greater i.e. W_{max} is taken as the bottleneck stage. Next the value of average workload on the basic stage is normalized to one unit and using the bottleneck criticality factor, g_3 which is defined as a factor to measure how much higher the workload is on a bottleneck machine, we calculate the target workload ratio on the bottleneck machine. Considering that the first stage, *x* is a basic stage and the second stage, *y* is the bottleneck machine, we calculate the target workload ratio on the bottleneck stage by the formula:

$$R_y = g_3 * (y - 1) + 1, \text{ where } y = 2. \quad (7.4)$$

In the case that, the first stage *(x)* is the bottleneck stage and the second stage *(y)* is the basic stage, the target workload ratio for the bottleneck stage *(x)* can be computed using the alternative formula:

$$R_x = g_3 * (y - x) + 1, \text{ where } x = 1 \text{ and } y = 2. \quad (7.5)$$

The processing times of the bottleneck stage have to be updated so that they meet their target workload ratio. Assuming that the second stage *(y)* is the bottleneck stage, we calculate the actual workload ratio by using the formula:

$$R`_y = \frac{W_y}{W_{min}}. \quad (7.6)$$

The processing times assigned to the jobs at the bottleneck stage are taken as $p^o_{y,i}$. They are now updated by setting them as follows:

$$p_{y,i} = p^0_{y,i} * \frac{R_y}{R`_y}. \quad (7.7)$$

This will ensure that value of the actual load ratio is the same:

$$\frac{W_y}{W_{min}} = R_y. \quad (7.8)$$

For the purpose of our testing, we calculate the average workload for both stages based on the maximum batch size and the number of machines on the stages. The stage with the larger workload ratio is assigned as the bottleneck stage and the procedure described above is used to adjust the processing times of the bottleneck stage. We use the value 0.25 for the factor g_3. The factor g_2 which is called the bottleneck location factor is used to indicate the stage at which the bottleneck exists.

The Table 7-1 below outlines how test cases will be produced to test the effectiveness of the proposed heuristics.

Table 7-1 Test data generation parameters

Problem Parameter	Values Used	Total values
Number of machines on Stage1	3, 5	2
Number of machines on Stage2	3, 5	2

Problem Parameter	Values Used	Total values
Number of jobs/family	10, 15, 25, 50	4
Batch size on stage 1	1,4, 8	3
Batch size on stage 2	1,4, 8	3
Number of families	3,5	2
Family processing time	5 with a probability of 0.2	1
	10 with a probability of 0.3	
	15 with a probability of 0.3	
	20 with a probability of 0.2	
Weight per job w_i	~ Uniform (0,1)	1
Release Dates for Stage1 ($r_{1,i}$)	~ Uniform $(0, \alpha*\sum(p_{1,i}/ mBf_{av})+(p_{2,i}/mBf_{av})$	2
	$\alpha = 0.25,0.75$	
Due dates for Stage2 ($d_{2,1}$)	$r_i + g_1*(p_{1,i} + p_{2,i})$	2
	$g_1 = 1.1, 1.5$	
Bottleneck Criticality Factor (G_3)	$g_3 = 0.25 , 0.375$	2
	Total Parameter Combinations	2304
	Number of problem instances/combination	5
	Total problem instances	11,520

From the above table, we can see that there are 10 variable parameters with two or more values for each. We run 5 replicates for each combination. This gives us a total of 11,520 problem instances.

Similar to Moench et al. (2005) [9], we fix the value of the time window Δt at 4 for the main experimentation because this provides a tradeoff for solution quality and time required for computation. Testing at different values of Δt is not included in the design of experiments but is tested at the values 4 and 8. Since the time window is only a property of the BATC and Iterative BATC heuristic, it is tested on those two heuristics.

Another important parameter that has not been included in the design of experiments is the look ahead parameter, k. The code has been designed so that it tests

the heuristic at different values of k starting from 0.5 and up to 5 in increments of 0.5. The value of the look ahead parameter from this grid which results in the solution with the least objective value, i.e. total weighted tardiness is chosen and that solution is reported.

Test instances are run on a 64 bit, Intel(R) Core™2 Due CPU, T6600 @ 2.20 GHz with a 4 GB RAM and Windows Vista software.

Chapter 8

COMPUTATIONAL RESULTS

The first stage of testing the quality of the solutions found by the heuristic involves comparison to the mathematical model for small problem instances. The mathematical model is unable to handle problems of size bigger than 3 families and 16 jobs/family and 2 machines per stage. The comparison of the heuristic against the mathematical model is done outside the design of experiments.

Table 8-1 below summarizes the average results of 10 problem instances of testing the solutions found by the BATC and Iterative BATC heuristic against the mathematical model for eight combinations of number of families and jobs/family. The results are represented in the form of value of total weighted tardiness/ratios compared against the best value, which is depicted in bold.

Table 8-1: Comparison of Mathematical Model to Heuristics

	Total Weighted Tardiness (Value/Comparison to best)		
Aggregate By:	**Mathematical Model**	**BATC Heuristic**	**Iterative BATC Heuristic**
2 families, 4 jobs/family	**48.3/1.00**	52.6/1.09	51.2/1.06
2 families, 8 jobs/family	**68.1/1.00**	75.2/1.10	72.9/1.07
2 families, 16 jobs/family	**78.4/1.00**	86.7/1.11	84.3/1.08
2 families, 30 jobs/family	**271.12/1.00**	318.5/1.17	316.1/1.17
3 families, 4 jobs/family	**57.3/1.00**	62.4/1.09	60.1/1.05
3 families, 8 jobs/family	**87.9/1.00**	97.7/1.11	95.2/1.08
3 families, 16 jobs/family	**121.6/1.00**	139.2/1.14	136.6/1.12
3 families, 30 jobs/family	**464.6/1.00**	547.8/1.18	540.6/1.16

As expected, the Iterative BATC heuristic performs better that the BATC Heuristic. However, it is important to note that the results of the heuristics are fairly close to each other. For small instances the heuristics provide solutions which at the worst are 18% worse than the optimal solution. This shows that the heuristics are capable of solutions of reasonable quality.

It is important to note that the MILP takes significantly longer than the BATC and the iterative BATC as can be seen by Table 8-2 below. Time required for computation jumps in when the number of jobs/family is beyond 16. In these two cases the MILP was allowed to run for 2 hours. MIP gap for these two instances is of interest and is calculated by using the formula, where LP (TWT) is the value of the LP relaxation:

$$1 - \frac{LP\ (TWT)}{MIP\ (TWT)} \tag{8.1}$$

The MIP gap observed was at an average of 0.39 (i.e. 39% for 2 families and 30 jobs/family) and 0.44 (i.e. 44% for 3 families and 30 jobs/family).

Table 8-2: Time taken for problem instances

	Time (Value in seconds/Comparison to best)		
Aggregate By:	**Mathematical Model**	**BATC Heuristic**	**Iterative BATC Heuristic**
2 families, 4 jobs/family	125/42	**3/1.00**	5/1.7
2 families, 8 jobs/family	140/47	**3/1.00**	5/1.7
2 families, 16 jobs/family	1660/332	**5/1.00**	6/1.2
2 families, 30 jobs/family	7200/720	**10/1.00**	15/1.5
3 families, 4 jobs/family	240/48	**5/1.00**	8/1.6
3 families, 8 jobs/family	350/70	**5/1.00**	8/1.6
3 families, 16 jobs/family	2990/498	**6/1.00**	10/1.6
3 families, 30 jobs/family	7200/600	**12/1.00**	15/1.25

Average results for the larger instances are presented in a table similar to design of experiments. In every case the Iterative BATC always yielded the best performance, while on an average the dispatching heuristic that came closest would be the ATC. The results are presented as values of total weighted tardiness/ratios against the solution of the best heuristic which is in bold.

8.1 Effect of number of machines and batch sizes:

The number of machines plays a crucial role in the value of the performance measure along with the stage in which the machines are located. It is seen that increasing the number of machines on the first stage reduces total weighted tardiness to a greater

extent as compared to increasing the number of machines on the second stage. This can be explained by examining the bottleneck effect that stage 1 plays when there are a fewer number of machines on it. As expected, it is seen that the case of 5 machines in both stages yields better results than when there are 5 machines on the first stage and 3 machines on the second. The case of 3 machines on the first stage and 5 on the second yields slightly poorer results, but still better than the case in which both stages have only 3 machines. The Iterative BATC heuristic achieves greater improvements followed by iterative BATC and ATC when the number of machines on a stage is increased. Table 8-3 summarizes these findings below.

Table 8-3: Effect of number of machines on each stage

	Total Weighted Tardiness (Value/Comparison to best)					
Aggregate By:	**By BATC**	**By Iterative BATC**	**By Random**	**By EDD**	**By Release Dates**	**By ATC**
3 m/c on Stage 1						
3 m/c on Stage 2	940.3/ 1.029	**913.6 / 1.000**	1565.4/ 1.713	1160.9 / 1.271	1158/ 1.267	994.7/ 1.088
5 m/c on Stage 2	845.7/ 1.043	**811.1/ 1.000**	1364.9/ 1.683	1127/ 1.39	1124.4/ 1.386	889.9/ 1.097
5 m/c on Stage 1						
3 m/c on Stage 2	631.4/ 1.050	**600.9/ 1.000**	975.9/ 1.624	759.1/ 1.263	757.1/ 1.26	719.4/ 1.197
5 m/c on Stage 2	529.7/ 1.064	**497.9/ 1.000**	786.4/ 1.579	625.1/ 1.255	623.5/ 1.252	585.6/ 1.176

It is seen that increasing the batch size from 4 jobs/batch to 8 jobs/batch greatly reduces the objective function, in most cases, with everything else being constant, increasing the batch sees up to a 40% reduction in total weighted tardiness. Results from problem instances grouped on the basis of batch sizes are presented below in Table 8-4.

Table 8-4 Effect of batch size

	Total Weighted Tardiness (Value/Comparison to best)					
Aggregate By:	**By BATC**	**By Iterative BATC**	**By Random**	**By EDD**	**By Release Dates**	**By ATC**
Size Stg 1: 1						
Size Stg 2: 1	1687/ 1.035	**1629.6/ 1.000**	2592.4/ 1.591	2060.8/ 1.265	2055.6/ 1.261	1799.7/ 1.104

	Total Weighted Tardiness (Value/Comparison to best)					
Aggregate By:	**By BATC**	**By Iterative BATC**	**By Random**	**By EDD**	**By Release Dates**	**By ATC**
Size Stg 2: 4	1417.1/ 1.033	**1371.3/ 1.000**	2177.6/ 1.588	1731.1/ 1.262	1726.7/ 1.259	1501.2/ 1.095
Size Stg 2: 8	1214.6/ 1.047	**1160.5/ 1.000**	1866.5/ 1.608	1483.8/ 1.279	1480/ 1.275	1277.3/ 1.101
Size Stg 1: 4						
Size Stg 2: 1	1012.2/ 1.039	**973.8/ 1.000**	1555.4/ 1.579	1236.5/ 1.270	1233.4/ 1.267	1069.4/ 1.098
Size Stg 2: 4	843.5/ 1.047	**805.8/ 1.000**	1296.2/ 1.609	1030.4/ 1.279	1027.8/ 1.275	878.8/ 1.091
Size Stg 2: 8	573.6/ 1.038	**552.5/ 1.000**	881.4/ 1.595	700.7/ 1.268	698.9/ 1.265	610.3/ 1.105
Size Stg 1: 8						
Size Stg 2: 1	472.4/ 1.052	**449.1/ 1.000**	725.9/ 1.616	577/ 1.285	575.6/ 1.282	486.4/ 1.083
Size Stg 2: 4	371.1/ 1.058	**350.7/ 1.000**	570.3/ 1.626	453.4/ 1.293	452.2/ 1.289	385.4/ 1.099
Size Stg 2: 8	269.9/ 1.070	**252.3/ 1.000**	420.8/ 1.668	342.7/ 1.358	328.9/ 1.303	297.5/ 1.179

As expected, the best solution is found when batch sizes and number of machines on both stages are set to the high values. The Iterative BATC Heuristic performs best in all cases. Also, it is seen that the Iterative BATC achieves greatest improvements in values of TWT when the batch size on a stage is increased.

8.2 Effect of G_1 (due date tightness factor) and α (release date factor):

The release date factor, α plays in important role in deciding how close release times and due dates are to each other. Smaller values of α yield release times which are clustered together. Smaller α values directly translate to jobs with due dates fairly close to each other. On pooling results based on α values, it is noticed that instances with lower α values, end up yielding higher total weighted tardiness. This seems logical, since in the real world this would translate to a greater number of jobs arriving very close to each other, causing a built up of inventory before the first stage. Given machine capacity constraints, this would cause some jobs to have long waiting times before they are processed.

Table 8-5 Effect of release date factor

	Total Weighted Tardiness (Value/Comparison to best)					
Aggregate By:	**By BATC**	**By Iterative BATC**	**By Random**	**By EDD**	**By Release Dates**	**By ATC**
Release Date Factor : 0.25	890.7/ 1.083	**822.1/ 1.000**	1491.9/ 1.815	1127/ 1.371	1110.4/ 1.351	909.9/ 1.106
Release Date Factor :0.75	624.6/ 1.060	**588.9/ 1.000**	994.9/ 1.689	769.1/ 1.305	787.1/ 1.336	640.4/ 1.087

On the other hand, larger α values mean jobs which have release dates and due dates which are further apart from each other. This allows machines to process jobs at a rate which is proportional to that of how they arrive. Hence, seeing better solutions for larger values of α seems logical. This can be seen by examining the values found in the table above which carry total weighted tardiness values for heuristics pooled by release date factor. It is seen that the BATC and ATC heuristic perform better in instances with ready times which are spread out i.e. α value is high.

Due dates have a significant bearing on the value of the total weighted tardiness. It is observed that when aggregating results on the basis of the due date factor, the best result is obtained when $G_1 = 1.5$. This implies that loose due dates will yield better solutions and that seems intuitive since we use a modified version of ATC, which, as a metric relies greatly on the amount of slack available on each job.

It is seen that the Iterative BATC and the BATC Heuristic perform best when we pool data based on due date tightness factor. The ATC Heuristic performs very well too since the ATC index largely depends on the due date and slack of a job. However, it is observed that relative to the Iterative BATC heuristic, the BATC and ATC heuristic tend to perform better in instances with loose due dates i.e. G_1 is set a higher value. Table 8-6 summarizes these findings.

Table 8-6 Effect of due date tightness factor

	Total Weighted Tardiness (Value/Comparison to best)					
Aggregate By:	**By BATC**	**By Iterative BATC**	**By Random**	**By EDD**	**By Release Dates**	**By ATC**
Duedate Tightness Factor : 1.1	836.1/ 1.037	**805.9/ 1.000**	1269.4/ 1.575	1009.1/ 1.252	1016.6/ 1.261	870.6/ 1.08
Duedate Tightness Factor : 1.5	650/ 1.058	**614.1/ 1.000**	987.3/ 1.608	746.9/ 1.216	775/ 1.262	655.7/ 1.068

8.3 Effect of number of jobs/family:

Increasing the number of jobs/family has direct bearing on the value of total weighted tardiness that the heuristics yield. Since greater number of jobs/family translates to greater test instances. Relative to the iterative BATC, the solution quality provided by the ATC heuristic seems to deteriorate with increasing number of jobs/family. Table 8-7 below summarizes this.

Table 8-7 Effect of number of jobs/family

	Total Weighted Tardiness (Value/Comparison to best)					
Aggregate By:	**By BATC**	**By Iterative BATC**	**By Random**	**By EDD**	**By Release Dates**	**By ATC**
Jobs/Family: 10	559.5/ 1.013	**551.9/ 1.000**	1055.4/ 1.912	678/ 1.228	692.7/ 1.255	581.2/ 1.053
Jobs/Family: 15	600.6/ 1.018	**590/ 1.000**	1073.5/ 1.819	806.1/ 1.366	777.6/ 1.318	664.6/ 1.126
Jobs/Family: 25	787.3/ 1.019	**772.7/ 1.000**	1297.5/ 1.679	942.1/ 1.219	955.8/ 1.237	892.1/ 1.154
Jobs/Family: 50	946.8/ 1.021	**927.3/ 1.000**	1637.1/ 1.765	1247/ 1.345	1196.9/ 1.291	1096.5/ 1.183

8.4 Effect of bottleneck criticality factor:

The results of the extensive testing carried out demonstrate that when g_3 is higher, the load on the bottleneck machine is higher which leads to higher values of total weighted tardiness. It is also seen that when the first stage is the bottleneck, the values of the objective function are higher. This can be explained by the fact that when the first stage is the bottleneck stage it acts as a siphon, preventing jobs from reaching the second

stage in a timely manner. Thus the value of the total weighted tardiness is greater when the first stage is the bottleneck stage.

Also, we noticed a direct correlation between the batch sizes and the bottleneck stage. As expected, the stage with the smaller batch size becomes the bottleneck stage. Table 8-8 below summarizes the findings when results are pooled on the basis of the bottleneck criticality factor. On examining the ratios of the results it becomes clear that the BATC and the Iterative BATC Heuristic perform best. The results of the BATC and the Iterative BATC are very close to each other. The next best heuristic is the ATC heuristic.

Table 8-8 Effect of bottleneck criticality factor.

	Total Weighted Tardiness (Value/Comparison to best)					
Aggregate By:	**By BATC**	**By Iterative BATC**	**By Random**	**By EDD**	**By Release Dates**	**By ATC**
Bottleneck Criticality Factor: 0.25	584.8/ 1.029	**568.4/ 1.000**	998.7/ 1.757	724.4/1. 275	712.6/ 1.254	642.1/ 1.130
Bottleneck Criticality Factor: 0.375	877.2/ 1.041	**842.6/ 1.000**	1458/ 1.73	1071.6/ 1.272	1068.9/ 1.269	958.1/ 1.137

8.5 Effect number of families:

In an effort to keep the number of instances at a reasonable number, only two variations of the number of families are included in the design of experiments. A greater number of families translate to a greater number of jobs in a problem instance as the total number of jobs is a function of number of families and number of jobs/family. Thus as expected when the number of families is increased the value of the objective function increases. Table 8-9 below gives the comparison of all heuristics when results are pooled on the basis of number of families.

Table 8-9 Effect of number of families

	Total Weighted Tardiness (Value/Comparison to best)					
Aggregate By:	**By BATC**	**By Iterative BATC**	**By Random**	**By EDD**	**By Release Dates**	**By ATC**
No. of families: 3	529.0/ 1.049	**504.5/ 1.000**	887.6/ 1.760	614.1/ 1.217	642.4/ 1.274	543.2/ 1.077
No. of families: 5	953.0/ 1.051	**906.5/ 1.000**	1349.1/ 1.488	1132/ 1.249	1169.1/ 1.289	997.0/ 1.100

Further testing on the number of families beyond the design of experiments is carried out where the number of jobs is set to 180 and the number of jobs/family becomes a function of 180/*f*, where *f* is the number of families. Experimental cases with 3, 5, 9 and 12 families were run using 5 machines on both stages and the batch size being set to 8. The release date factor (α) was set to 0.75, the due date tightness factor (g_1) to 1.5 and the bottleneck criticality factor (g3) to 0.25.Table 8-10 shows the results of experimentation.

Table 8-10 Effect of number of families outside of experimentation

	Total Weighted Tardiness (Value/Comparison to best)					
Aggregate By:	**By BATC**	**By Iterative BATC**	**By Random**	**By EDD**	**By Release Dates**	**By ATC**
No. of families: 3	1005.2/ 1.049	**958.5/ 1.000**	1686.4/ 1.760	1204.7/ 1.257	1258.6/ 1.313	1132.1/ 1.181
No. of families: 5	1320.2/ 1.040	**1269.2/ 1.000**	1888.8/ 1.488	1584.8/ 1.249	1608.7/ 1.268	1467.8/ 1.156
No. of families: 9	1163.9/ 1.039	**1119.8/ 1.000**	1952.7/ 1.744	1394.9/ 1.246	1457.4/ 1.301	1295.1/ 1.151
No. of families: 12	1461.7/ 1.041	**1405.1/ 1.000**	2091.2/ 1.488	1724.6/ 1.227	1781.1/ 1.267	1514.4/ 1.077

As seen from the table above the BATC and Iterative BATC heuristic yield relatively better results when the number of families is restricted to smaller numbers. When the number of families is increased, the solution quality decreases, but Iterative BATC still provides better solution quality than others. Thus it is most important to use the proposed heuristics when the number of incompatible families is lower. ATC

provides reasonable results when the number of families is large. For the f = 12 case, the improvement on solution quality of the Iterative BATC and the BATC heuristic becomes less remarked when compared to the ATC.

8.6 Effect of varying time window:

Experimentation for two values of the time window (Δt) are carried out. Larger values of the time window allow a greater number of jobs to be considered each time a machine becomes available which leads to a greater number of possibilities in the number of batch combinations that are feasible and to be considered. Due to this there is a slight increase in computational time as expected. Table 8-11 below represents the effect of varying the time window. It is seen that The Iterative BATC performs better than all other heuristics.

Table 8-11 Effect of varying the time window

	Total Weighted Tardiness (Value/Comparison to best)					
Aggregate By:	**By BATC**	**By Iterative BATC**	**By Random**	**By EDD**	**By Release Dates**	**By ATC**
Δt : 4	848.7/ 1.053	**805.9/ 1.000**	1274.2/ 1.581	1018.8 /1.264	1034.2/ 1.283	905.3/ 1.123
Δt : 8	624.8/ 1.067	**585.3/ 1.000**	1274.2/ 2.177	1018.8/ 1.741	1034.2/ 1.767	905.3/ 1.547

8.7 BATC v/s Iterative BATC:

As expected, based on the testing carried out it becomes clear that the Iterative BATC Heuristic performs slightly better than the BATC heuristic in every case. However, this is only true when multiple passes of the Iterative BATC are carried out. Upon further consideration, the reason why the iterative BATC does not yield better results when only one pass is carried out because it highly depends on the weight and slack of each job. In some cases assigning the start time of stage 2 as the due date for stage 1 would lead to a more loose due date for that job on stage 1, which in turn would

reduce its ATC index on account of the greater slack now available. In such cases, this job would now be processed later as compared to the earlier case, which would cause an increase in the total weighted tardiness for the system.

In general, using the Iterative BATC proves to be a better approach if time is not a constraint since the Iterative BATC approach takes a slightly longer time to carry out especially in those instances involving a greater number of jobs.

Chapter 9

CONCLUSIONS AND FUTURE WORK

9.1 Conclusions

From the experimentation conducted, it seems that the BATC and the iterative BATC heuristics both perform pretty well. It can be said that it would be prudent to use the Iterative BATC in those cases where due dates of jobs were relatively loose and the jobs had greater slack. This would allow the Iterative BATC to improve on the solution by being able to reassign calculated due dates for the first stage as start times for the second.

However, it must also be mentioned that the Iterative BATC provides only a slight improvement on the final solution but the computational time required as compared to the BATC heuristic is slightly more. In extremely large problem instances, it could be argued, that the BATC would be a practically feasible option as compared to the Iterative BATC. In general it is noticed that the lowest value of the objective function is reached by the third pass in most cases and the fourth pass in some cases.

The proposed heuristics can also be used to solve a special case of the problem at hand where ready times of all jobs are set to 0, i.e. all jobs are available for processing at time t = 0. Thus the problem at hand can be simplified to FF2|batch incompatible|$\sum w_i T_i$. Since, all jobs are available for processing at the time the heuristic is run, the subset of jobs that can be used to calculate the next batch to be processed will consist of all unprocessed jobs from the family to be processed. Thus the heuristic will force all batches to run at capacity as that will ensure that the solution has the smallest total weighted tardiness. Another point worth noting is that, the length of the time window, Δt can be set to ∞ to reach the same result as all jobs are already available at time t = 0 and the heuristic does not need to wait for additional jobs to become ready for processing.

9.2 Future Work

Based on findings from testing done on the two heuristics, certain areas of further research which would make it more usable in real environments seem promising. The environment considered in this research approximates set up times to be zero. It is assumed that machines do not need any set up and all different families are processed as soon as they are scheduled on machines. A further extension to our research can be to include family and stage dependent set up times for both stages, which will make its real world implications greater since on a factory floor, each different type of job normally requires some sort of setup on the machine to customize the machine for the processing required.

Furthermore, this research assumes that there is unlimited buffer space between the two stages. However, this is not necessarily true in all practical situations, where because of space constraints there might be a limited buffer space between the two stages. It could be an interesting area to investigate how solutions would change when the number of jobs that can wait in front of the second stage is restricted. The assumptions of Klemmt et. al. (2009) [12] such as machine specific batch sizes and dedication of machines make the problem more realistic or real world like.

Another extension to this research could be to test the developed heuristics with wider environments, additional stages or even for other performance measures such as makespan.

REFERENCES

1. T. C. E. Cheng, T. Y. Kovolou & K. N. Chakhlevich. Batching in a two-stage flow-shop with dedicated machines in the second stage. IIE Transactions 2004 36, 87–93.

2. Y. Kim, B. Joo & J. Shin. Heuristics for a two stage hybrid flow-shop scheduling problem with ready times and a product-mix ratio constraint. J Heuristics 2009 15, 19–42.

3. C. Liao & L. Liao. Improved MILP models for two-machine flow-shop with batch processing machines. Mathematical and Computer Modeling 48 (2008) 1254–1264.

4. T. Cheng, T., Z. Chen, M. Kovalyov, & B. Lin. Parallel-machine batching and scheduling to minimize total completion time. IIE Transactions, 1996 28, 953.

5. A. Oulamara. Makespan minimization in a no-wait flow shop problem with two batching machines. Computers & Operations Research 34 (2007) 1033–1050.

6. N. Hall, G. Laporte, E. Selvarajah & C. Sriskandarajah. Scheduling and Lot Streaming in Flow-shops with No-Wait in Process. IIE Transactions (2002) 34, 953–970.

7. P. Brucker, A. Gladky, J.A. Hoogeveen, M.Y. Kovalyov, C.N. Potts, T. Tautenhahn, S.L. van de Velde. Scheduling a batching machine. Journal of Scheduling (1998) 31 - 54.

8. L. Tang & P. Liu. Minimizing makespan in a two-machine flow-shop scheduling with batching and release times. Mathematical and Computer Modeling 49 (2009) 1071-1077.

9. L. Moench, H. Balasubramanian, J. W. Fowler, M. E. Pfund. Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. Computers & Operations Research 32 (2005) 2731–2750.

10. H. Balasubramanian, L. Moench, J. W. Fowler, M. E., Pfund. Genetic algorithm based scheduling of parallel batch machines with incompatible families to minimize total weighted tardiness. International Journal of Production Research 42 (2004) 1621–38.

11. L. Moench, H. Balasubramanian, J. W. Fowler, M. E., Pfund. Minimizing total weighted tardiness on parallel batch processing machines using genetic algorithms. Proceedings of the International Symposium on Operations Research, Klagenfurt, Austria;2002.p.205–11.

12. A. Klemmt, C. Almeder, L. Moench, G. Weigert. A comparison of MIP based decomposition techniques and VNS approaches for batch scheduling problems. IIE Transactions (2009) 1686 – 1694.

13. A. Devpura. Scheduling Parallel and single batch machines to minimize total weighted tardiness. Doctoral Dissertation (Arizona State University), June 2003.

14. A. P. J. Vepsalainen, T. E. Morton. Priority rules for job shops with weighted tardiness costs. Management Science 1987; 33(8):1035–47.

15. Y. Yang, S. Kreipl and M. Pinedo. Heuristics for minimizing total weighted tardiness in flexible flow shops. Journal of Scheduling. J. Sched. 2000; 3:89-108

16. D. C. Montgomery. Design and Analysis of Engineering Experiments. Published by John Wiley & Sons (2008).

17. R Graham, E Lawler, J Lenstra, A Rinnooy Kann. Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics 1979; 5:287–326.

APPENDIX A

MATHEMATICAL MODEL CODE

```
/*********************************************
 * OPL 6.1 Model
 * Author: Anubha
 * Creation Date: Feb 28, 2011 at 7:41:39 PM
 *********************************************/
int job = ...;
int batch = ...;
int machine_in_1 = ...;
int machine_in_2 = ...;
int family = ...;
int stage = ...;
int G = ...;
range I=1..job;
range J=1..batch;
range K1=1..machine_in_1;
range K2=1..machine_in_2;
range L=1..family;
range ST=1..stage;
;
float d[I]=...;
float w[I]=...;
float r[I]=...;
float p[ST][I]=...;
int B[ST]=...;
int a[L][I]=...;
;
dvar boolean x[I][J][K1][ST];
dvar boolean y[J][K1][L][ST];
;
dvar float+ s[1..(batch+1)][K1];
dvar float+ t[1..(batch+1)][K2];
dvar float+ C[ST][I];
dvar float+ T[I];
;
minimize sum(i in I) (w[i]*T[i]);
;
subject to
{
forall (i in I, st in ST)
A1:   sum(j in J, k1 in K1) (x[i][j][k1][st]) == 1;
;
forall (j in J, k1 in K1, st in ST)
A2:   sum(i in I) (x[i][j][k1][st]) <= (B[st]);
;
forall (j in J, k1 in K1, st in ST)
A3:   sum(l in L) (y[j][k1][l][st]) == 1;
;
forall (i in I, j in J, k1 in K1, l in L, st in ST)
A4:   (y[j][k1][l][st]) - (a[l][i])*(x[i][j][k1][st]) >= 0;
;
forall (i in I, j in J, k1 in K1, st in ST)
A5:   (x[i][j][k1][st])*(r[i]) <= (s[j][k1][st]);
;
forall (i in I, j in J, k1 in K1, st in ST)
A6:   (s[j][k1][st]) + (p[st][i])*(x[i][j][k1][st]) <=
(s[j+1][k1][st]);
```

```
;
forall (i in I, j in J, k1 in K1, st in ST)
A7:   (G*(1 - (x[i][j][k1][st]))) + (C[st][i]) >= (s[j][k1][st]) +
(p[st][i]);
;
forall (i in I, j in J, k1 in K1, k2 in K2)
A8:   (C[1][i]) <= (s[j][k2][2]) + G*(1 - (x[i][j][k2][2]));
;
forall (i in I)
A9:   (C[2][i]) - (T[i]) <= (d[i]);
};
execute DISPLAY_RESULTS{
  writeln("Tardiness =",T);
  writeln("Completion Time =",C[2]);
  writeln("x =",x);
  writeln("u =",u);
  writeln("Stage 1 Start Times =",s);
  writeln("Stage 2 Start Times =",t);
}
```