Multi-Criteria Flow-Shop Scheduling Optimization

A Senior Project Submitted

In Partial Fulfillment

Of the Requirements for the Degree of

Bachelor of Science in Industrial Engineering

Presented to:

The Faculty of California Polytechnic State University,

San Luis Obispo

By:

Teja Arikati

Project Advisors: Reza Pouraghabagher & Karla Carichner

Graded By: _____ Date of Submission: _____.

Checked by: _____ Approved by: _____.

Acknowledgements

I would like to thank all my professors over the years for giving me the opportunity to grow and learn at Cal Poly. I would personally like to thank my project advisors Dr. Pouraghabagher and Karla Carichner for advising me during this project. Also, my interest in Operations Research was stemmed from Dr. Freed's classes and I would like to thank her for helping me realize my passion for Operations Research.

Table of Contents

Abstract 1

Introduction 2

Background/Literature Review 3

Mixed Integer Linear Programming Formulation 6

Design Requirements 9

Implementation 11

Results 14

Conclusion 17

References 19

Appendix 20

Abstract

A flow-shop is a type of manufacturing job shop where similar jobs follow a similar, linear sequence through the shop. Every day, flow-shops receive several different orders and it is up to the scheduler to plan the daily schedule. This schedule should be designed to prevent bottlenecks in the shop, to have on-time delivery of products, and satisfy several other requirements. Often times, schedulers perform subjective scheduling and utilize simple heuristics or just intuition to schedule the jobs. With computer-based scheduling, schedulers can now create schedules and determine quantitatively what sorts of schedules work best. Currently, much of the computer-based schedules only try to optimize for one KPI such as Total Tardiness.

This paper considers incorporating multiple-criteria into computer based scheduling so that schedulers can have more flexibility and develop schedules which optimize multiple-criteria; this paper specifically considers minimizing Total Tardiness and maximizing Throughput. Comparisons between single-criterion models and the multiple-criteria model are made and it is discovered the multiple-criteria model provides a great compromise in optimizing both KPIs. A user-friendly program is developed where schedulers of any flow-shop can utilize the software to compute schedules for cases up to 10 jobs and 10 machines.

Introduction

Flow-shops are manufacturing shops where similar products or jobs follow a specific set of processing steps and the parts made are generally quite similar. Also, all the jobs follow a linear order where they don't go and revisit a previous machine for processing. This is different from a job-shop where a job can follow any order among the machines and can even visit a machine multiple times. Job scheduling is a huge problem which every flow-shop company faces. When orders come into a flow-shop, the main role of the scheduler is to schedule the jobs so that the jobs are built on-time and as fast as possible. Many flow-shops do this manually. Below is an example of a flow-shop compared to a job-shop:

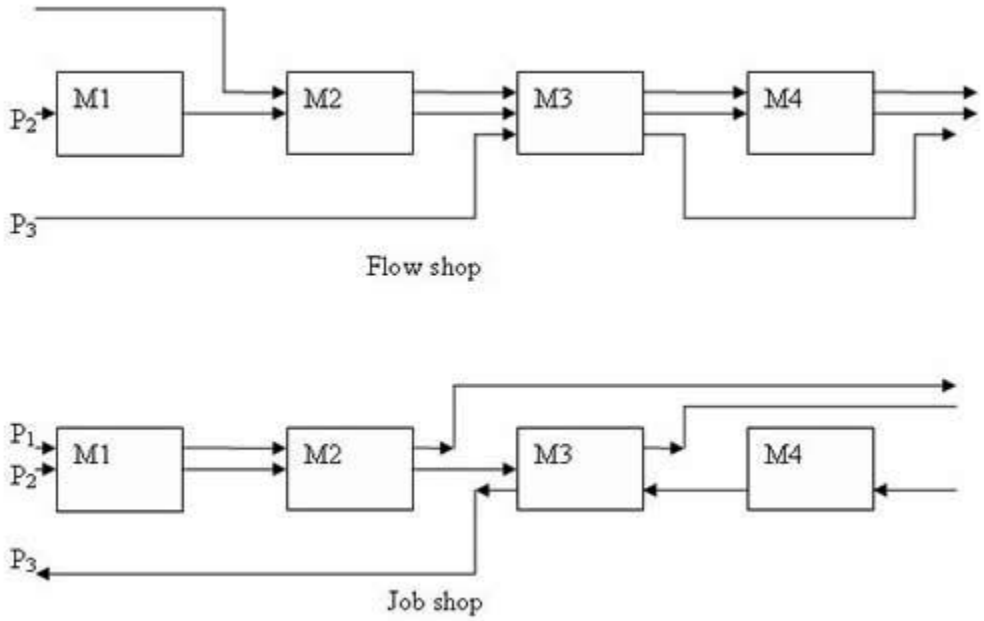

Figure 1: Comparison between Flow-shop and Job-Shop

Since the jobs follow a linear fashion in a flow-shop, the scheduling of jobs in flow-shops tends to be simpler than for jobs in job-shops. For this project, the scope will be focused only on flow-shop scheduling.

The goal of this project is to develop a general-purpose software application which will provide a manufacturing company with the optimal flow-shop schedule. This general-purpose software application should work with any flow-shop company. What an “optimal” flow-shop schedule may vary day to day therefore a multi-criteria formulation is created where schedulers can set their preferences for different KPIs. A comparison between a single criterion-formulation and the multi-criteria formulation will be conducted to determine the effectiveness of the new model.

Background/Literature Review

As stated above in the introduction, flow-shop scheduling is a difficult venture which every flow-shop tries to tackle. The complexity of flow-shop scheduling falls in the NP hard classification (Hanen 1994). What this means is that it takes exponentially more effort to schedule jobs as more jobs are added. For this reason, several heuristics and other methods are used to tackle this problem. Literature for flow-shop scheduling exploded after the introduction of a heuristic algorithm by S.M. Johnson which is now called Johnson’s Algorithm (Johnson 1954). This algorithm uses a list of rules to develop a flow-shop schedule that optimizes for total elapsed time. For two machines, this algorithm provides a schedule with the minimum Total Elapsed time but for schedules with more than two machines, this algorithm doesn’t guarantee the best solution. 70 years after the formulation of the Johnson’s Algorithm, the depth of research into flow-shop scheduling is lackluster. Only over the last 20 years, with the explosion of computing power, have researchers started considering more innovative ways of solving this problem. Several different heuristics have been developed for flow-

shop scheduling and these methods include genetic algorithms and ant-colony optimization (Pezzella 2008). Most of the research, however, focuses on single-criterion optimization and the research that does cover multi-criteria only covers bi-criteria for machine counts less than 3 (Dhingra 2010). The purpose of this paper is to fill in the research and implementation gap for multi-criteria flow-shop scheduling.

Looking at flow-shop scheduling formulations, several different Mixed-Integer Linear Programming (MILP) formulations are found in literature. In the literature, three distinct formulations appear. These three different formulations described by Wagner (Wagner 1959), Wilson (Wilson 1989), and Manne (Manne 1960) are listed under Appendix A, B, C respectively. To quickly go over the three formulations:

1. The first formulation by Wagner revolves around the relationships between Idle Times, Wait Times, and Processing times of consecutive jobs between consecutive machines.
2. The second formulation by Wilson revolves around the relationships between Start Times and Processing times of consecutive jobs between consecutive machines.
3. The third formulation by Manne revolves around the relationships between Completion Times, Processing times, and precedence requirements of consecutive jobs between consecutive machines.

In a paper by Ronconi and Birgin (Ronconi & Birgin 2012), the computation times for all three methods were measured by solving sample cases which varied in number of jobs and number of machines. These formulations were run through a Simplex Solver which is the same algorithm used by Microsoft Excel's solver. The results showed

similar results between Wagner's and Wilson's formulations with Wilson's formulation being slightly faster. Manne's formulation was significantly slower and even infeasible for large sample sizes (15 jobs and 10 machines). The exact run times are listed in Appendix D.

Modeling multi-criteria objectives could be done in several ways. Two methods will be discussed in this literature review. The first method is through applying a weighted average to each objective directly. Each objective is included in the objective function and a weight from 0-1 is applied to each objective (Dhingra 2010). The sum of all the weights must equal 1. This methodology follows the same principles as weighted averages and other "weighted" calculations. A disadvantage of this method is that it is difficult to gauge the actual "weight" given to each objective because of their different ranges. What is meant by this is that each of these objectives has values which lie in different ranges and different magnitudes. For example: Tardiness could range from 0 days to 20 hours, Total Elapsed Time from 100 hours to 200 hours, and Total Flow Time from 300 hours to 500 hours. If equal weighting was applied to each of these objectives, Total Flow Time would have higher priority compared to TET or Tardiness because it has a larger range than the other two KPIs. To combat this, there must be a way to normalize every objective so that the desired priority is applied to each objective.

The way to normalize objectives is through the application of fuzzy set theory. Fuzzy set theory is a part of set theory and was introduced by Lotfi Zadeh (Zadeh 1965). The purpose of fuzzy set theory is to apply a continuous gradient to generally discrete constraints. This paper will be looking at the normalization part of fuzzy set theory. To normalize the objectives, the range of each of the objectives must be known.

Using the previous example, the general range for tardiness is [0,20] hours, for TET is [100,200] hours, and for TFT is [300,500] hours. To normalize each objective, we would divide each objective by their total range. By doing this, each objective is effectively transformed to a value between 0 and 1. Now, the weighted average method can be utilized to combine these objectives into an objective function. A great example of fuzzy set theory application in scheduling is given by Sima Rokni (2010). The formulation is shown in Appendix E.

Mixed Integer Linear Programming Formulation

Based on the literature review, 3 popular formulations exist for flow-shop scheduling. 2 of these formulations, the ones by Wagner and Wilson, are more efficient and feasible. Wagner's and Wilson's formulations proved to perform quite similarly. Below is a table comparing both algorithms by listing the total number of constraints, total binary variables, total continuous variables, complexity of code, and computational run-times.

	# of Constraints	# of Binary Variables	# of Continuous Variables	Ease of Implementation	Computational Run-times
Wagner	nm + 3n - 1	n^2	nm + 3n - 1	Medium-High	Great
Wilson	2nm + 3n - m	n^2	2nm + 3n - m	Medium	Great

Table 1: Comparison between Wagner's and Wilson's Formulation

Although Wilson's formulation would generally have more continuous variables and number of constraints than Wagner's, the code complexity required to implement Wilson's algorithm would be simpler than Wagner's. Since the performance of both

algorithms is similar, the easier implementation of Wilson's formulation was chosen for the foundation of the model presented in this report.

Variables & inputs of the following model with their corresponding bound constraints are listed below:

$x_{ij}\ is\ binary\ where\ x_{ij} \in \{0,1\},$

$T_j \geq 0, \qquad C_{jm}, \qquad S_{jk}$

$i = 1, \dots, n, \qquad j = 1, \dots, n, \qquad k = 1, \dots, m, \qquad p_{ik}, \qquad d_i$

What each of these variables means are as follows: x_{ij} is equal to 1 if job i (there are n number of jobs and these jobs are labelled between 1 and n) is in the j-th position of the sequence, equal to 0 otherwise. T_j is the Tardiness of the j-th job in the sequence (remember here that this is not related to i which is the original label of the jobs, the j only corresponds to the sequence order). C_{jm} corresponds to the completion of the j-th job of the sequence at machine m (the last machine). S_{jk} corresponds to the start time of the j-th job at machine k. i is ordered from 1 to n where n is the number of jobs in the flow-shop. j is ordered from 1 to n where n is the number of jobs in the flow-shop (the length of the sequence is the same as the number of jobs). k is ordered from 1 to m where m is the number of machines in the flow-shop. p_{ik} is the processing time of job i at machine k. d_i is the due date of job i.

The following formulation is for the single-criterion flow-shop model which aims to minimize total Tardiness of the jobs:

$$Minimize \sum_{j=1}^{n} T_j$$

$$subject\ to$$

$$T_j \geq C_{jm} - \sum_{i=1}^{n} x_{ij} d_i, \qquad for\ all\ j$$

$$C_{jm} = S_{jm} + \sum_{i=1}^{n} x_{ij} p_{im}, \qquad for\ all\ j$$

$$S_{j+1,k} \geq S_{jk} + \sum_{i=1}^{n} x_{ij} p_{ik}, \quad for\ all\ j\ except\ n\ and\ for\ all\ k$$

$$S_{j,k+1} \geq S_{jk} + \sum_{i=1}^{n} x_{ij} p_{ik}, \quad for\ all\ j\ and\ for\ all\ k\ except\ m$$

$$S_{11} \geq 0$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad for\ all\ j$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad for\ all\ i$$

The single-criterion option for maximizing throughput requires just changing the objective function as follows:

$$Minimize \sum_{j=1}^{n} C_{jm}$$

The multi-criteria option which optimizes for both throughput and tardiness requires the addition of fuzzy constraints as discussed before during the literature review. What this requires is the addition of two user inputs $f_\alpha\ and\ f_\beta$ which are the normalizing factors which will effectively convert two factors into a range between 0 and

1. Also two more inputs indicating importance must be included and these inputs will be labelled as $P_\alpha \; and \; P_\beta$. The constraints will remain the same as before from the single-constraint formulation. The revised objective function is as follows:

$$Minimize \sum_{j=1}^{n} \left(\frac{P_\alpha * C_{jm}}{f_\alpha} + \frac{P_\beta * T_j}{f_\beta} \right)$$

Design Requirements

Program Input Requirements

Since there is no specific customer in this project, the parameters and constraints were set for ease and practicality so that virtually any flow-shop could use the program. The basic expected inputs from the flow-shop will be as follows.

1. **Jobs & Machines**

   A list of jobs that need to be completed is provided. This should also include the number of jobs and the number of machines in the flow-shop.

2. **Manufacturing Run Times:**

   The manufacturing lead time for each of these jobs including wait time, queue time, setup time, and run time should be provided. However, a run time for each job at each machine is sufficient if other specific data is not available.

3. **KPIs or Goals to Achieve:**

   The scheduler needs to input what KPIs or goals are to be optimized as well as a ranked priority for each of these. For example, a scheduler could choose to minimize Tardiness and maximize Throughput with ranks of 1 and 2 respectively.

This means the program will try to achieve the Tardiness goal with a higher priority compared to the Throughput goal.

User Interface Requirements:

The program will have a clean and intuitive layout which a tester can easily interface with. This will be presented through a Python interactive interface. The scheduler can then type the mentioned inputs above through this application. There will be an option to run random jobs through a flow-shop or to input actual processing times. After running the program, a clean and clear output of the job schedule should be displayed. Another key aspect to the User Interface will be clear graphical outputs of the results. Bar graphs and statistical comparisons will be displayed to show the differences between various program designs (single-criterion vs multi-criteria).

Technical Aspects/Backend Design Requirements:

This program will be built from ground up in Python. There will be two programs built; the first will be an implementation of the single-criterion flow-shop scheduling program and the second will be an implementation of the multi-criteria flow-shop scheduling program. Wilson's formulation proved to be one of the fastest computationally and the easiest to implement therefore this formulation will be used. For the multi-criteria design, the use of fuzzy constraints will be incorporated so that the schedulers can accurately assign priorities to different KPIs. The optimization add-on, which will be the workhorse for computing the results, is Gurobi Optimization Version 7. This software package is industry leading in terms of having the fastest run-times for

Mixed Integer Linear Programming Formulations. A free academic license was acquired for Gurobi.

Implementation

In this section, the back-end code used to incorporate the constraints and the objectives will be described. The entire code is in Appendix F. Following this explanation, an overview of the User Interface will be described.

Code

Firstly, all the variables need to be initialized. The binary x_{ij} variables, the continuous start time variables S_{jk}, and the continuous Tardiness variables T_j are initialized in the code below:

```
#Create num_jobs^2 binary constraints, num_jobs tardiness constraints, and num_jobs*num_machines start time constraints
for i in range(num_jobs):
    for j in range (num_jobs):
        vars_bin[i,j] = m.addVar(vtype = GRB.BINARY, name = 'BIN_POS' +str(i) +'JOB' +str(j))
    for j in range(num_machines):
        vars_start[i,j] = m.addVar(vtype = GRB.CONTINUOUS,name = 'START_J' +str(i) + 'M' +str(j))
    vars_tardi[i] = m.addVar(vtype = GRB.CONTINUOUS, name = 'Tardiness of Job: ' + str(i))
m.update()
```

Figure 2: Code for variable initialization

Secondly, tardiness and completion time constraints need to be specified. These are the lines in the formulation: $T_j \geq C_{jm} - \sum_{i=1}^{n} x_{ij} d_i,\ for\ all\ j;\ C_{jm} = S_{jm} + \sum_{i=1}^{n} x_{ij} p_{im},\ for\ all\ j$. The code is listed below:

```
#Tardiness Constraint
for n in range(num_jobs):
    dd_sum = sum([vars_bin[m,n]*time_to_commit[m] for m in range (num_jobs)])
    m.addConstr(vars_tardi[n]>=0)
    m.addConstr(vars_tardi[n]-vars_start[n,num_machines-1]+dd_sum>=0)
```

Figure 3: Code for tardiness and completion time constraints

Thirdly, the start time constraints need to be specified. These are the lines in the formulation: $S_{j+1,k} \geq S_{jk} + \sum_{i=1}^{n} x_{ij} p_{ik}$, $for\ all\ j\ except\ n\ and\ for\ all\ k$; $S_{j,k+1} \geq S_{jk} + \sum_{i=1}^{n} x_{ij} p_{ik}$, $for\ all\ j\ and\ for\ all\ k\ except\ m$; $S_{11} \geq 0$. The code is listed below:

```
#Initial Start Time Constraint
m.addConstr(vars_start[0,0] >=0)

#num_jobs*(num_machines-1) constraints for flowshop
for i in range(num_jobs):
    for j in range(num_machines-1):
        const_1 = 0
        for n in range(num_jobs):
            const_1 = const_1 + vars_bin[i,n]*routing_file[j,n]
        m.addConstr(vars_start[i,j+1] - vars_start[i,j] - const_1 >= 0)
m.update()

#(num_jobs-1)*num_machines constraints for flowshop
for i in range(num_jobs-1):
    for j in range(num_machines):
        const_2 = 0
        for n in range(num_jobs):
            const_2 = const_2 + vars_bin[i,n]*routing_file[j,n]
        m.addConstr(vars_start[i+1,j] - vars_start[i,j] - const_2 >= 0)
m.update()
```

Figure 4: Code for start time constraints

Fourthly, the binary constraints need to be specified. These are the corresponding lines in the formulation: $\sum_{i=1}^{n} x_{ij} = 1,\ for\ all\ j$; $\sum_{j=1}^{n} x_{ij} = 1,\ for\ all\ i$. The code is listed below:

```
#Every Position has 1 Job Constraint
for n in range(num_jobs):
    z = 0
    for x in range(num_jobs):
        z = z + vars_bin[n,x]
    m.addConstr(z == 1)
#Every Job has 1 Position Constraint
for n in range(num_jobs):
    z = 0
    for x in range(num_jobs):
        z = z + vars_bin[x,n]
    m.addConstr(z ==1)
```

Figure 5: Code for binary constraints

Lastly, the objective functions need to be defined. These three objective functions are defined: $Minimize\ \sum_{j=1}^{n} T_j$; $Minimize\ \sum_{j=1}^{n} C_{jm}$; $Minimize\ \sum_{j=1}^{n} \left(\frac{P_\alpha * C_{jm}}{f_\alpha} + \frac{P_\beta * T_j}{f_\beta} \right)$. The code is listed below:

```
sum([vars_bin[num_jobs-1,n]*routing_file[num_machines-1,n] for n in range(num_jobs)])
obj_final_processing = [sum([vars_bin[x,n]*routing_file[num_machines-1,n] for n in range(num_jobs)]) for x in range(num_jobs)]


obj_function_TET = vars_start[num_jobs-1, num_machines-1] + obj_final_processing[num_jobs-1]
obj_function_flow = sum([vars_start[x,num_machines-1] + obj_final_processing[x] for x in range(num_jobs)])

obj_function_proc = .5*obj_function_TET + .5*obj_function_flow
obj_function_TTC = 0
for x in range(num_jobs):
    obj_function_TTC += vars_tardi[x]

obj_function_mult = a*obj_function_flow + b*obj_function_TTC


#obj_function optimizing
if option ==1:
    m.setObjective(obj_function_proc,GRB.MINIMIZE)
    m.optimize()

elif option ==2:
    m.setObjective(obj_function_TTC,GRB.MINIMIZE)
    m.optimize()
elif option ==3:
    m.setObjective(obj_function_mult, GRB.MINIMIZE)
    m.optimize()
```

Figure 6: Code for Objective function definitions

User Interface

In the User Interface, the user can change the number of jobs to any number from 1 to 5 and the number of machines to any number from 1 to 5. There is also a choice to choose which scenario (objective) function to run. The first two scenarios are the single-criterion objective functions while the third scenario is the multi-criteria objective function. After this, the throughput and total tardiness of the flow-shop is displayed. A graph displaying the difference in throughput and tardiness of the chosen scenario to the other two scenarios is also visualized at the bottom.

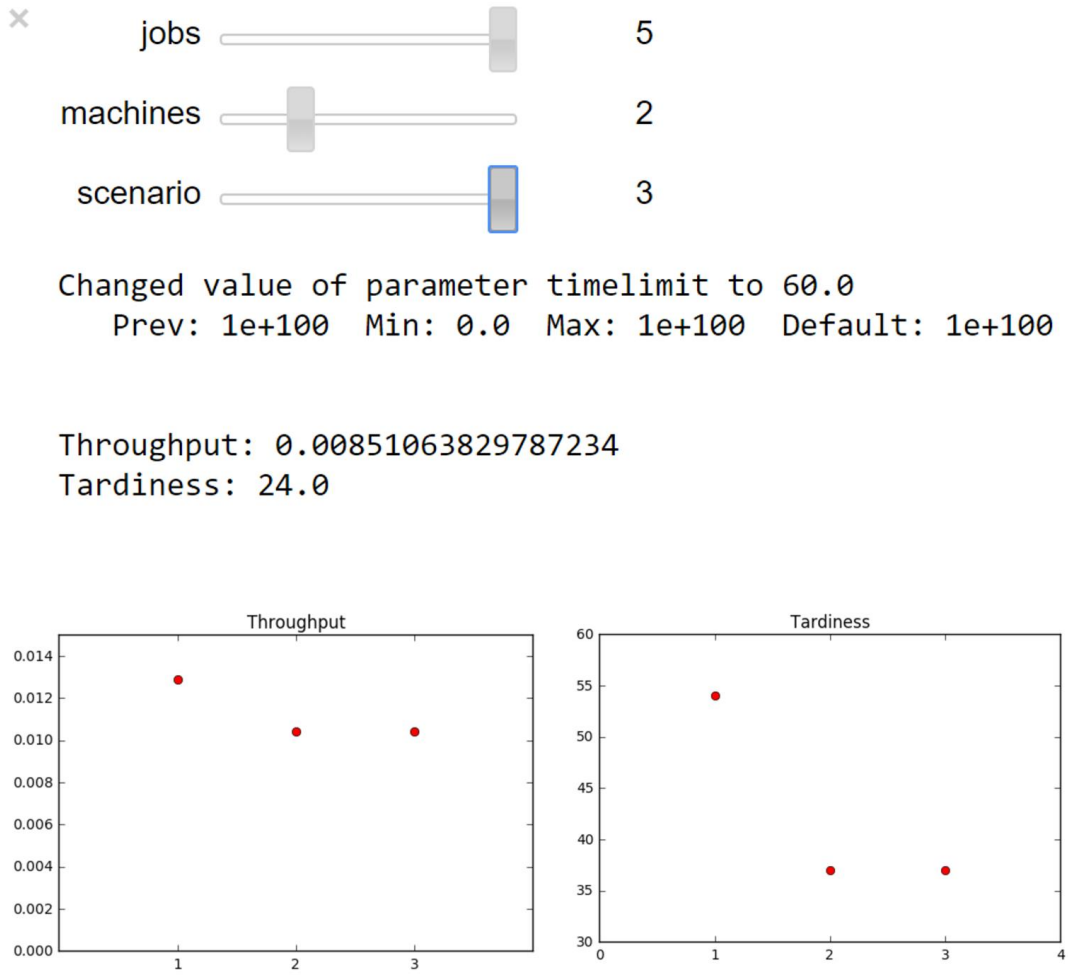

Figure 7: Sample User Interface

Results

To test the effectiveness of each of the scenarios, an experiment was conducted comparing each scenario's tardiness and throughput results. In this experiment, the flow-shop was scheduled to have 10 jobs with each of these jobs needing to be processed at 10 machines. The run times of each of these jobs at each of the machines was randomly created by the computer following a uniform distribution between 4 and 16. The due dates of each of these jobs were randomly created by the computer

following a uniform distribution between 90 and 200. The throughput and tardiness results were recorded for each of the scenarios. The experiment was run 20 times by the computer and all the data was stored into Python arrays. The experiment took 10 minutes to complete on an Intel i5-6300U CPU.

Based on this experiment, the average tardiness and average throughput of each of the scenarios are as follows:

	Average Tardiness	Average Throughput
Single-Criterion: Minimize Tardiness	80.65	.001110
Single-Criterion: Maximize Throughput	189.6	.001160
Multi-Criteria: Optimize Both Criteria	96.05	.001146

Table 2: Average Tardiness and Average Throughput of each Model

Looking at the data, it is necessary to perform statistical analyses to determine significance. To compare the values, one-way ANOVAs were performed in Python to compare tardiness values and throughput values. Conducting ANOVA to compare the tardiness values and throughput values among the programs resulted in a p-value of .0000144 for tardiness and a p-value of .0038 for throughput. This means at a 95% confidence level; the values of tardiness and throughput are statistically different.

To visualize the data sets, bar graphs and box-plots are presented below comparing the tardiness and throughput performance of each of the three models:

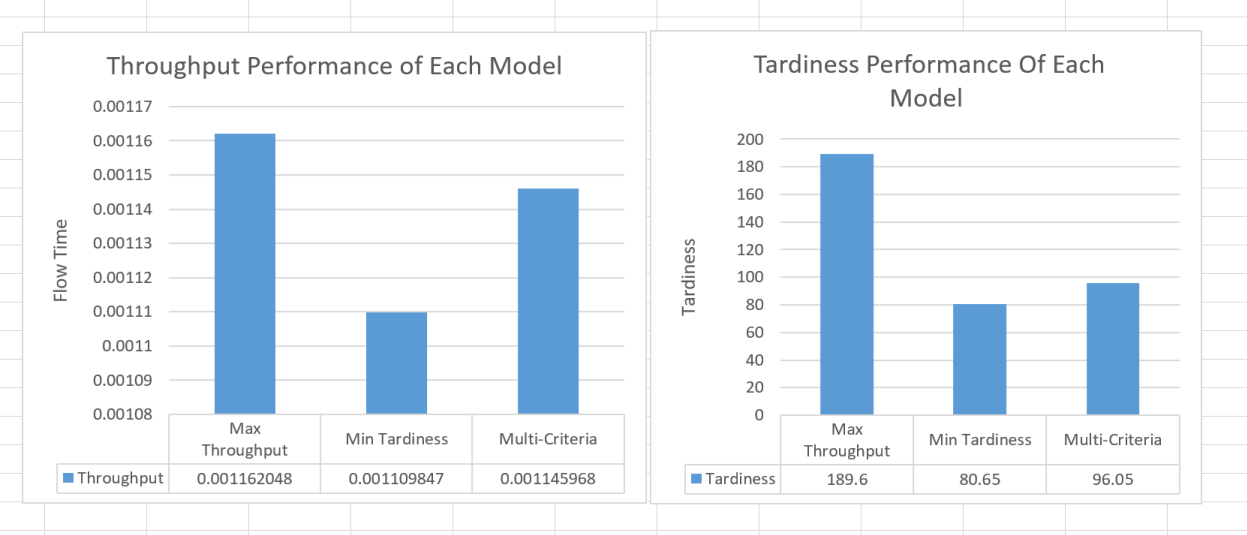

Figure 8: Bar graphs comparing Throughputs and Tardiness of each Model

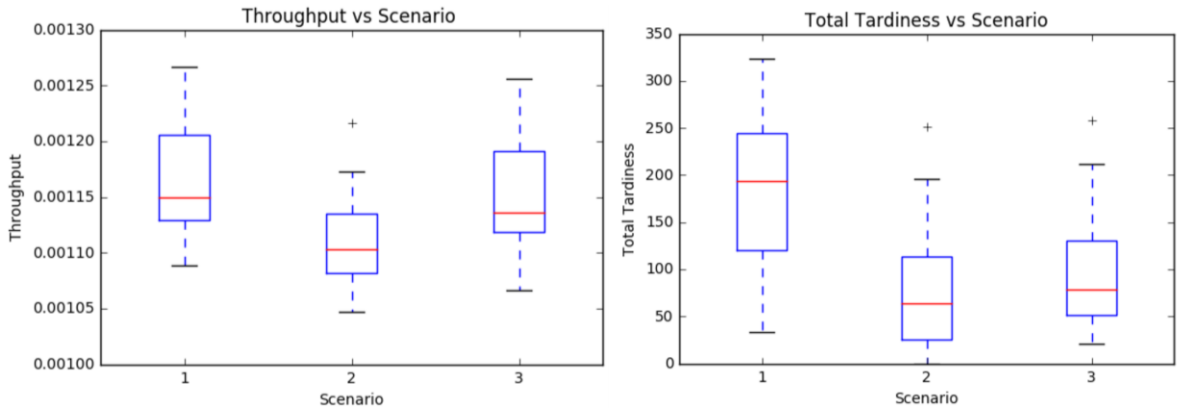

Figure 9: Box plots comparing Throughputs and Tardiness of each Model

Based on the results, there is a **3.2%** greater throughput with the multi-criteria model compared to the single-criterion tardiness model. Also there is a **49.3%** improvement in Total Tardiness with the multi-criteria model compared with the single-criterion throughput model. The multi-criteria model, as would be expected, resulted in total tardiness and throughput values in between corresponding tardiness and throughput values in both single-criterion models. This can be visually seen in the bar

graphs and the box-plots. Sample run cases with different machine and job counts with all three different scenarios display similar results to the results found with the 20 run experiment conducted with 10 jobs and 10 machines.

Conclusions

The results from the experiment prove that the multi-criteria program performs as planned. The multi-criteria program proved to provide solutions which were compromises between the single-criterion models. The feasibility of the multi-criteria program was of concern but all programs finished within 20 seconds for even 10 job and 10 machine scenarios. This program proves to become infeasible as machine and job counts go past 20 where run times take an hour or more but even the single-criterion program started to become infeasible at these scales.

Since there was no industry sponsor, it was difficult to find a way to do an economic analysis. If an industry sponsor was available for this project, comparisons of throughput and tardiness between subjective scheduling and computer-based scheduling could be made. Based on the time saved, an economic analysis could be conducted to see how much money could be saved.

Scheduling is a problem that flow-shops face on a daily problem. With various priorities, such as customer deadlines and production efficiency, it can be difficult to develop a schedule which can satisfy every goal and objective. As discussed in this paper, computer-based scheduling can help schedulers create schedules that satisfy different KPI requirements. With the inclusion of multi-criteria objectives and the development of a user-friendly interface for schedulers to interact with, the proposed program in this paper offers a simplistic tool that schedulers can utilize to create more

optimum schedules. The requirements each day may change and perhaps priorities for each different KPI may change as well. Having low tardiness may be important one day while having high throughput may be important another day. The ability to alter priorities gives schedulers higher flexibility and flexibility is extremely important in a flow-shop where requirements can change at a moment's notice.

References:

1. A. Dhingra, P. Chandna, "Multi-objective flow-shop scheduling using hybrid simulated annealing", Measuring Business Excellence, Vol. 14 Iss: 3 (2010), pp.30 – 41
2. C. Hanen. "Study of a NP-hard cyclic scheduling problem: The recurrent job-shop." European journal of operational research 72.1 (1994): 82-101.
3. S.M. Johnson, “Optimal two- and three-stage production schedules with setup times included”, Naval Res. Log. Quart. I (1954) 61-68.
4. A. S. Manne, “On the job-shop scheduling problem”, Operations Research 8 (1960), pp. 219–223,
5. F. Pezzella, G. Morganti, G. Ciaschetti, “A genetic algorithm for the Flexible Job-shop Scheduling Problem”, Computers & Operations Research, Volume 35, Issue 10 (2008), Pages 3202-3212
6. S. Rokni “Optimization of industrial shop scheduling using simulation and fuzzy logic”, University of Alberta M.S. Thesis (2010)
7. D. Ronconi, E. Birgin, “Mixed-integer programming models for flowshop scheduling problems minimizing the total earliness and tardiness”**,** Just-in-Time Systems (2012), pp. 91-105
8. H. M. Wagner, “An integer linear-programming model for machine scheduling, Naval Research Logistic 6 (1959), pp. 131–140,
9. J. M. Wilson, “Alternative formulations of a flow-shop scheduling problem”, Journal of the Operational Research Society 40 (1989), pp. 395–399,
10. L. A. Zadeh . “Fuzzy sets”. Inform. Contr. 8 (1965):338-53,

Appendix

A. Wagner's MILP Formulation for Flow-Shop Scheduling

$$\text{Minimize} \sum_{j=1}^{n} E_j + T_j$$

$$\text{subject to } T_j \geq C_{jm} - \sum_{i=1}^{n} x_{ij} d_i, \qquad j = 1, \ldots, n,$$

$$E_j \geq \sum_{i=1}^{n} x_{ij} d_i - C_{jm}, \qquad j = 1, \ldots, n,$$

$$C_{1m} = \sum_{k=1}^{n-1} (\sum_{i=1}^{n} x_{i1} p_{ik} + W_{1k}) + \sum_{i=1}^{n} x_{i1} p_{im},$$

$$C_{jm} = C_{j-1,m} + I_{j-1,m} + \sum_{i=1}^{n} x_{ij} p_{im}, \qquad j = 2, \ldots, m,$$

$$I_{jk} + \sum_{i=1}^{n} x_{i,j+1} p_{ik} + W_{j+1,k} = W_{jk} + \sum_{i=1}^{n} x_{ij} p_{i,k+1} + I_{j,k+1}, \qquad j = 1, \ldots, n-1,$$

$$k = 1, \ldots, m-1,$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1, \ldots, n,$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1, \ldots, n.$$

B. Wilson's MILP Formulation for Flow-Shop Scheduling

$$\text{Minimize } \sum_{j=1}^{n} E_j + T_j$$

$$\begin{aligned}
\text{subject to } & T_j \geq C_{jm} - \sum_{i=1}^{n} x_{ij} d_i, && j = 1, \ldots, n, \\
& E_j \geq \sum_{i=1}^{n} x_{ij} d_i - C_{jm}, && j = 1, \ldots, n, \\
& C_{jm} = S_{jm} + \sum_{i=1}^{n} x_{ij} p_{im}, && j = 1, \ldots, m, \\
& S_{j+1,k} \geq S_{jk} + \sum_{i=1}^{n} x_{ij} p_{ik}, && j = 1, \ldots, n-1, k = 1, \ldots, m, \\
& S_{j,k+1} \geq S_{jk} + \sum_{i=1}^{n} x_{ij} p_{ik}, && j = 1, \ldots, n, k = 1, \ldots, m-1, \\
& S_{11} \geq 0, \\
& \sum_{i=1}^{n} x_{ij} = 1, && j = 1, \ldots, n, \\
& \sum_{j=1}^{n} x_{ij} = 1, && i = 1, \ldots, n.
\end{aligned}$$

C. Manne's MILP Formulation for Flow-Shop Scheduling

$$\text{Minimize } \sum_{i=1}^{n} E_i + T_i$$

$$\begin{aligned}
\text{subject to } & T_i \geq C_{im} - d_i, && i = 1, \ldots, n, \\
& E_i \geq d_i - C_{im}, && i = 1, \ldots, n, \\
& C_{i1} \geq p_{i1}, && i = 1, \ldots, n, \\
& C_{ik} \geq p_{ik} + C_{i,k-1}, && i = 1, \ldots, n, k = 2, \ldots, m, \\
& C_{ik} \geq p_{ik} + C_{jk} - M z_{ij}, && i = 1, \ldots, n-1, j = i+1, \ldots, n, k = 1, \ldots, m, \\
& C_{jk} \geq p_{jk} + C_{ik} - M(1 - z_{ij}), && i = 1, \ldots, n-1, j = i+1, \ldots, n, k = 1, \ldots, m.
\end{aligned}$$

D. Run Times for each formulation. (MUB1,MUB2,MUB3 are Wagner, Wilson, and Manne Respectively)

		Model MUB1			Model MUB2		
n	m	CPU Time	Simplex It	B&B nodes	CPU Time	Simplex It	B&B nodes
5	3	0.03	131.03	14.17	0.02	111.89	10.33
10	3	0.61	7,517.03	533.08	0.49	7,341.31	497.03
10	7	1.94	25,943.47	1,082.28	1.65	23,295.50	1,093.39
10	10	4.59	66,336.97	2,065.44	3.65	51,774.64	2,037.33
15	3	10.81	194,452.58	8,230.92	9.63	159,715.39	7,343.39
15	7	91.25	1,501,181.78	34,280.94	75.50	1,207,768.81	36,477.86
15	10	281.25	4,506,249.72	86,486.28	227.75	3,566,364.53	91,435.28
20	3	105.90	1,719,383.14	68,800.81	89.59	1,456,790.72	54,698.81

		Model MUB3			Model MUB4		
n	m	CPU Time	Simplex It	B&B nodes	CPU Time	Simplex It	B&B nodes
5	3	0.05	182.31	22.17	0.05	331.08	33.17
10	3	2.11	37,891.92	2,956.08	3.28	58,925.89	5,075.83
10	7	27.90	316,057.22	28,688.97	39.45	461,485.92	44,965.47
10	10	40.17	417,254.75	32,041.25	61.87	635,131.11	50,661.58

E. Fuzzy Constraint Formulation

$$\mu_q(D_i) = \begin{cases} 1 & f_q \le f_q^* \\ \dfrac{f_q^H - f_q}{f_q^H - f_q^*} & f_q^* \le f_q \le f_q^H \\ 0 & f_q \le f_q^H \end{cases}$$

$$\mu_q(D_i) = \begin{cases} 0 & f_q \le f_q^H \\ \dfrac{f_q - f_q^H}{f_q^* - f_q^H} & f_q^H \le f_q \le f_q^* \\ 1 & f_q \le f_q^* \end{cases}$$

F. Complete Back-End Code

```python
import numpy as np
import pandas as pd
import math
from gurobipy import *

def scheduling_criteria(routing_file, time_to_commit, option,weights=(.5,.5),fuzzy=(1,1)):
    num_machines, num_jobs = routing_file.shape
    a = weights[0]
    b = weights[1]
    fuz_a = fuzzy[0]
    fuz_b = fuzzy[1]

    m = Model("Single Criteria")
    m.params.timelimit = 60
    m.params.OutputFlag = 0
    vars_bin = {}
    vars_start = {}
    vars_tardi = {}
    #Create num_jobs^2 binary constraints, num_jobs tardiness constraints, and num_jobs*num_machines start time constraints
    for i in range(num_jobs):
        for j in range (num_jobs):
            vars_bin[i,j] = m.addVar(vtype = GRB.BINARY, name = 'BIN_POS' +str(i) +'JOB' +str(j))
        for j in range(num_machines):
            vars_start[i,j] = m.addVar(vtype = GRB.CONTINUOUS,name = 'START_J' +str(i) + 'M' +str(j))
        vars_tardi[i] = m.addVar(vtype = GRB.CONTINUOUS, name = 'Tardiness of Job: ' + str(i))
    m.update()
    #Tardiness Constraint
    for n in range(num_jobs):
        dd_sum = sum([vars_bin[m,n]*time_to_commit[m] for m in range (num_jobs)])
        m.addConstr(vars_tardi[n]>=0)
        m.addConstr(vars_tardi[n]-vars_start[n,num_machines-1]+dd_sum>=0)
    #Every Position has 1 Job Constraint
    for n in range(num_jobs):
        z = 0
        for x in range(num_jobs):
            z = z + vars_bin[n,x]
        m.addConstr(z == 1)
    #Every Job has 1 Position Constraint
    for n in range(num_jobs):
        z = 0
        for x in range(num_jobs):
            z = z + vars_bin[x,n]
        m.addConstr(z ==1)

    #Initial Start Time Constraint
    m.addConstr(vars_start[0,0] >=0)

    #num_jobs*(num_machines-1) constraints for flowshop
    for i in range(num_jobs):
        for j in range(num_machines-1):
            const_1 = 0
            for n in range(num_jobs):
                const_1 = const_1 + vars_bin[i,n]*routing_file[j,n]
            m.addConstr(vars_start[i,j+1] - vars_start[i,j] - const_1 >= 0)
    m.update()
```

```
    #(num_jobs-1)*num_machines constraints for flowshop
    for i in range(num_jobs-1):
        for j in range(num_machines):
            const_2 = 0
            for n in range(num_jobs):
                const_2 = const_2 + vars_bin[i,n]*routing_file[j,n]
            m.addConstr(vars_start[i+1,j] - vars_start[i,j] - const_2 >= 0)
    m.update()
    #OBJ_FUNCTION TO MINIMIZE FLOW TIME
    sum([vars_bin[num_jobs-1,n]*routing_file[num_machines-1,n] for n in range(num_jobs)])
    obj_final_processing = [sum([vars_bin[x,n]*routing_file[num_machines-1,n] for n in range(num_jobs)]) for x in range(num_jobs)]


    obj_function_TET = vars_start[num_jobs-1, num_machines-1] + obj_final_processing[num_jobs-1]
    obj_function_flow = sum([vars_start[x,num_machines-1] + obj_final_processing[x] for x in range(num_jobs)])

    obj_function_proc = .5*obj_function_TET + .5*obj_function_flow
    obj_function_TTC = 0
    for x in range(num_jobs):
        obj_function_TTC += vars_tardi[x]

    obj_function_mult = a*obj_function_flow + b*obj_function_TTC


    #obj_function optimizing
    if option ==1:
        m.setObjective(obj_function_proc,GRB.MINIMIZE)
        m.optimize()

    elif option ==2:
        m.setObjective(obj_function_TTC,GRB.MINIMIZE)
        m.optimize()
    elif option ==3:
        m.setObjective(obj_function_mult, GRB.MINIMIZE)
        m.optimize()

    final_processing = [sum([vars_bin[x,n].X*routing_file[num_machines-1,n] for n in range(num_jobs)]) for x in range(num_jobs)]
    #for x in range(num_jobs):
     #   for y in range(num_jobs):
      #      print(str(int(vars_bin[x,y].X)) + "   ", end='' )
       # print ()
    """
    for x in range(num_jobs):
        print(str(round(vars_start[x,num_machines-1].X)) + "  ", end='')
    print()
    print('Tardiness Values')
    print([round(vars_tardi[m].X) for m in range(num_jobs)])
    print()
    print (time_to_commit)
    print ([round(vars_start[x,num_machines-1].X) for x in range(num_jobs)])

    #print ([routing_file[num_machines-1,n] for n in range(num_jobs)])
    print(sum([vars_tardi[m].X for m in range(num_jobs)]))
    print(obj_function_proc.getValue())
    """"""
    print ('\n')
    print ("Throughput: " + str(1/obj_function_proc.getValue()))
    print ("Tardiness: " + str(obj_function_TTC.getValue()))
    print ()
    return (obj_function_proc.getValue(), obj_function_TTC.getValue())
```