2019

Pharmaceutical Scheduling Using Simulated Annealing And Steepest Descent Method

Bryant Jamison Spencer
bspence3@mix.wvu.edu

PHARMACEUTICAL SCHEDULING USING SIMULATED ANNEALING AND STEEPEST DESCENT METHOD

Bryant J. Spencer

Problem Report submitted to the
Statler College of Engineering and Mineral Resources
at West Virginia University

in partial fulfillment of the requirements for the degree of

Master of Science
in
Industrial Engineering

Alan McKendall, PhD, Committee Chairperson
Robert Creese, PhD
Ashish Nimbarte, PhD

Department of Industrial and Management Systems Engineering

Morgantown, West Virginia
2019

Keywords: Pharmaceutical Manufacturing, Scheduling, Flexible Flowshop, Simulated Annealing, Steepest Descent Method

Abstract

PHARMACEUTICAL SCHEDULING USING SIMULATED ANNEALING AND STEEPEST DESCENT METHOD

Bryant J. Spencer

In the pharmaceutical manufacturing world, a deadline could be the difference between losing a multimillion-dollar contract or extending it. This, among many other reasons, is why good scheduling methods are vital. This problem report addresses Flexible Flowshop (FF) scheduling using Simulated Annealing (SA) in conjunction with the Steepest Descent heuristic (SD).

FF is a generalized version of the flowshop problem, where each product goes through *S* number of stages, where each stage has *M* number of machines. As opposed to a normal flowshop problem, all 'jobs' do not have to flow in the same sequence from stage to stage. The SA metaheuristic is a global optimization method for solving hard combinatorial optimization problems. SD is a local search method that keeps track only of the current solution and moves only to neighboring permutations based on the largest decrease in the objective function value. The goal of this problem report is to use FF in conjunction with SA to minimize the makespan (length of schedule) in a pharmaceutical manufacturing environment. There are 4 total stages in the tentative production route: granulation, compression, coating, and packaging. This process will be uniform; as in, each stage will have the same number of identical machines.

In this study, SA solved the illustrative small-scale example problems precisely and efficiently using a very small amount of computation time. Afterward, the SD heuristic is used to ensure that the best solution found by SA is a local optimum. SD did not improve upon the solutions found by SA.

Dedication

Dedicated to my wife, Tonya Spencer, my children, Jacobi Spencer, Kiara and Avery Gibson, my parents, Franki and Gerard Spencer, and my brothers, Donavan and Sebastian Spencer. In memory of my Nana, Mary Elizabeth Taylor.

Keep the family close.

Acknowledgments

I am very grateful to my advisor, Dr. Alan McKendall, who assisted in my admittance to the master's program and went out of his way to ensure that I finished. Without his help and guidance over the past several years, I may not have finished my research. I would like to thank Dr. Robert Creese, who took time out of his busy schedule to be a part of my committee, even in retirement. Thank you to Dr. Ashish Nimbarte for being a part of my committee and ensuring that I could finish out this last semester to graduate. I would also like to thank Marie Owen for always being able to help any time that I needed it.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Manufacturers are the source (suppliers) of the prescription drugs in the pharmaceutical supply chain. The pharmaceutical manufacturing industry is composed of two distinct business models: manufacturers of brand-name drugs (e.g., Pfizer, Merck, and Novartis) and manufacturers of generic drugs (e.g., Mylan, Roxane, and Barr) (Health Strategies Consultancy, 2005). The major difference is that brand-name manufacturers allocate most resources toward research and development of new drugs, while generic manufacturers formulate drugs directly based on a branded version with an expired patent.

Pharmaceutical manufacturing for tablets utilizes four generalized steps (Fig. 1): granulation, compression, coating, and packaging.

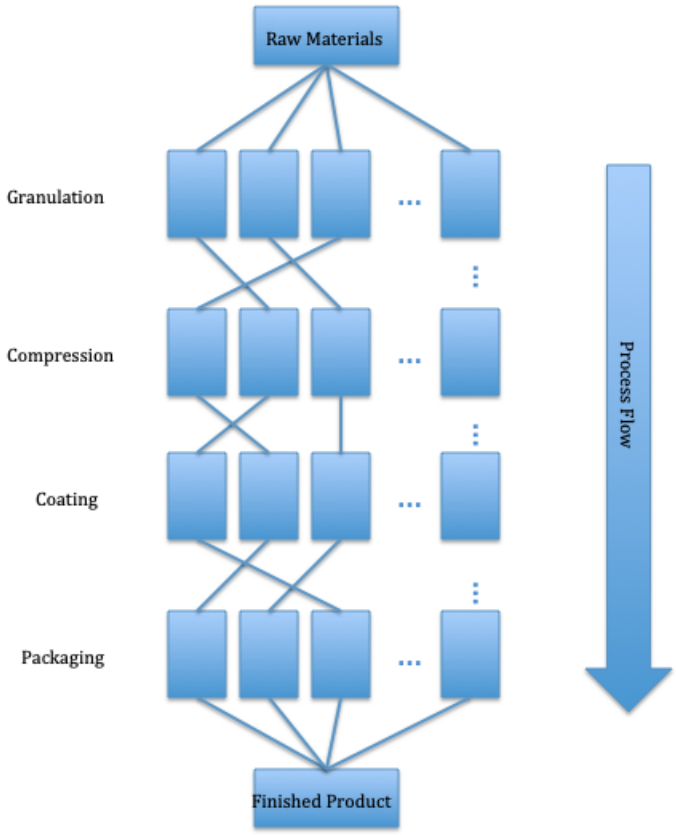

Figure I: Four Steps in Pharmaceutical Manufacturing

Raw materials are received, checked for quality, and assigned lot numbers. The active and inactive ingredients are then processed and blended to the appropriate consistency. The blending operation

typically uses v-blenders. Raw materials are rotated in these blenders to achieve a homogenous blend. Then wet or dry granulation techniques are used to further process the ingredients. The wet granulation process consists of three-story tall fluid bed granulators and high shear mixers. The dry powder is suspended in mid air by high volume airflow. A granulating solution is sprayed onto the suspended powder. The temperature is then increased to dry the wet granulation. This powder is then milled to achieve a specific particle size and density. The powder is then blended again for uniformity.

At the compression stage, the powder is then compressed into tablets or filled into capsules. For the purpose of the study, only tablets will be focused on. A tablet press consists of a large, rotating turret, where upper and lower punches are forced together within a die to form tablets. Tablets are then placed into coating pans. A coating machine resembles an industrial clothes dryer; however, a coating machine also has a spray gun system in order to coat the tablets at desired temperature. The tablets are then coated with a color or clear coating. The coating can serve as decorative (for identification) or functional (to hide taste, extended release, etc.). These coated tablets are then packaged into bottles.

Since the pharmaceutical manufacturing industry is so competitive, one of the most, if not the most, important drivers is time-to-market. Time-to-market is the time it takes from a product being conceived until it's processed and available for sale. Pharmaceutical is also a highly regulated industry, with large amounts of cleaning, set-ups, and trainings. There are other major factors, including, but not limited to: high but uncertain demand, long lead times from suppliers, and different product variations (different milligrams, extended release, delayed release, chewable).

These are some of the many reasons why scheduling is so important in the pharmaceutical manufacturing industry. It helps to minimize production time while fulfilling predicted and actual demand. The overall objective would be to minimize late deliveries; late deliveries lead to penalties and loss of contracts. In this problem report a four-stage scheduling problem is considered.

Chapter 2: Literature Review

Combinatorial approach

Combinatorial approaches are based on the changing of one permutation (solution) to another by switching jobs around in order to optimize a given objective function (Sethanan, 2001). The process searches for an optimal value whose solution space is a discrete but large configuration space. Some simple examples of typical combinatorial optimization problems are:

- Traveling Salesman Problem
- Bin-Packing
- Job-shop Scheduling

Enumerative optimal methods

The most general techniques are mathematical formulations (such as linear programming, dynamic programming, integer programming), and branch and bound methods (Sethanan, 2001). The complexity of the variable interactions makes these methods far too difficult and time consuming.

Heuristic approach

Exact solution measures may not exist or may be too exhaustive to apply for large-sized, or even small sized, scheduling problems (Sethanan, 2001). It is then necessary to use heuristics, which will yield good solutions. These solutions may or may not be optimal, but they supply a local optima that is acceptable. The following are examples of heuristic approaches:

1. Relaxed Exact Solutions
    a. Linear Programming relaxation
    b. Lagrangian relaxation
2. Local Search Techniques
    a. Random Descent
    b. Steepest Descent
3. Meta-Heuristics
    a. Tabu Search
    b. Simulated Annealing
    c. Genetic Algorithms
4. Ad Hoc Decision Rules

Simulated Annealing

Simulated annealing (SA) is derived from an analogy between the physical annealing of solids and combinatorial optimization. Physically, it refers to the heating of a substance close to the melting point, staying at this temperature for a set time, and then lowering the temperature slowly until the substance reaches a stable state. This process softens the substance by removing internal stresses, but the substance never actually leaves the solid state. Metropolis et al. (1953) realized that Markov chains, a random sequence of states whose probabilities depend on the previous state, could be used to converge to a probability distribution. Furthermore, Kirkpatrick (1983) discovered that there is a deep, useful connection between statistical methods and combinatorial optimization. A detailed analogy with annealing solids provides a framework for optimization of very large, complex systems (Kirkpatrick, 1983).

Simple Flowshop Problem

Johnson (1954) developed an unrestricted flowshop (FS) where each item is to be produced on machine one and then machine two. A simple decision rule is obtained in the literature for the optimal scheduling of the production so that the makespan (total length of schedule) is a minimum (Johnson, 1952). A restricted three-stage problem was also explored.

Flexible Flowshop w/o Restriction

Flexible flowshop (FFS) is the generalized FS problem. Where FS is a specific number of machines in a series, FFS is a series of stages with a parallel amount of machines per stage (Shieh, 2004).

Two-stage FFS

Arthanari et al. (1971) presented a branch and bound algorithm to optimally solve the special case of the two-stage flexible flowshop (FFS) where there are multiple machines at the first stage and only one machine at the second (Crowder, 2006). Gupta (1988) addressed the two stage FFS problem where there are identical machines at each stage. A heuristic was developed for a special case where there is only one machine at stage two. This problem has also been examined by Blazewicz et al. (1992) to show that Johnson's algorithm (1954) and the longest processing time (LPT) rule can be proven to be the best and closest to optimal (Shieh, 2004). Chen (1995) also developed a heuristic for the special case with one machine at the second stage. Koulamas (2000) considered two-stage and three-stage FFS with parallel machines at each stage. The objective was to minimize makespan and was accomplished using lower complexity algorithms. Soewandi (2001)

successfully developed several heuristic procedures of time O(n log n) to solve the three-stage FFS.

Multiple stage (>2 stages) FFS with unlimited buffer

Brah and Hunsucker (1991) developed a branch and bound algorithm to solve scheduling problems that optimize maximum completion time for facilities. The lower bounds and elimination rules developed are based upon the generalization of the flow shop problem. Brockman et al. (1997) improved Brah and Hunsucker's (1991) algorithm; however, their algorithm was not able to handle availability time of machines until 1998. Portman (1998) is also an improved algorithm to Brah and Hunsucker (1991). It was proved that the original lower bound may decrease along a path of the search tree, and genetic algorithms was used to improve the search value of the upper bound. A problem with fifteen jobs and five stages was solved with a 3% deviation from the branch and bound method. Verma and Dessouky (1998) present a branch and bound procedure which provides an optimal solution to the 3-stage problem, and a fast heuristic procedure that is shown to provide good approximate solutions on sample problems. This heuristic is a natural extension of the 2-stage polynomial-time procedure. Verma and Dessouky (1999) compared their results/algorithm to the Latest Start Time rule (LST), which refers to the latest time at which the activity can be completed without delaying the project.

Multiple stage (>2 stages) FFS with buffer limitation (machine blockage)

Buffer limitation, or machine blockage, is where a completed job may remain on a machine and occupy it until a downstream machine becomes available. Gilmore (1964) found the minimal cost sequence with a special case traveling salesman problem, where time or money is used for changing over a machine for the next job. Salavador (1973) solved this type of problem using branch & bound algorithm. Wittrock (1988) solved this problem by minimizing the makespan and queueing time. The problem is decomposed into three subproblems and each of these is solved using a fast heuristic. The algorithm was tested by computing schedules for a real production line. Sawik (2000) presents new mixed integer programming formulations for scheduling of a flexible flow line with blocking. The basic mixed integer programming formulations have been enhanced to model blocking scheduling with alternative processing routes where for each product a set of routes is available for processing, and a reentrant flow line where a product visits a set of stages more than once is also considered (Sawki, 2000). Sawik (2002) also develops a mixed integer programming approach for lines that consist of finite intermediate buffers, which cause machine blocking.

Chapter 3: Problem statement and objectives of research

This study is based on the need of a pharmaceutical company to schedule its production process. This pharmaceutical company manufactures a vast array of products in a flexible flow shop environment. A flow shop is a problem that has a set amount of products that each go through a set amount of machines, as opposed to job shop, where each product has its own route through a set of machines. Flexible flow shop is an even more general form of flow shop where each product goes through one of many identical machines at each stage. Both of these instances are NP-hard and generally considered too complex to solve using exact methods.

Because scheduling problems have a vast amount of variation, the following assumptions are made for the problem under consideration (assumptions noted with an asterisk have been deemed realistic)

1. **N* products, *S* stages, *M* identical machines per stage.
2. *Preplanned campaign based; campaign contains smaller lots or batches of the same product family. Product families contain similar variation of a product (different milligram, delayed release, extended release, chewable). This will minimize set-up and cleaning times that arise when switching between product families (Guomundsdóttir, 2012).
3. *Production time is per campaign.
4. Each stage will have the same number of machines.
5. *The number of jobs to be scheduled and their processing times on each machine at each stage is known in advance and fixed.
6. *The number of stages and the machine configuration at each stage are known in advance and fixed.
7. Preemption, temporarily interrupting a job, is not allowed. Once a job has started, it must be completely finished on the assigned machine before it can move to the next stage.
8. All jobs are ready to begin processing at time period 0.
9. Jobs may or may not be scheduled in the same order at each stage, i.e. job passing is allowed.
10. *Set-up and clean times are included in the processing time of each job at each stage.
11. Buffers/storage space is ignored. It is unlimited.
12. Machine blocking cannot occur, so when a job is finished processing, it can leave the machine before there is room on a machine at the next stage
13. There are no due dates associated with the jobs and the objective is to minimize the makespan.

14. *Production is considered as make-for-stock based on a forecast, as opposed to make-to-order.
15. *Transportation time between stages and buffers are considered to be negligible.

There are five main objectives to this research:

1. Consider the main objective of minimizing the makespan for the multi-stage flexible flowshop with uniform machines and unlimited buffers.
2. Use a random permutation as the initial job processing order. Construction heuristics used in conjunction with a random model produce no advantage.
3. Since this problem is considered strongly NP-Hard, use SA to solve this problem.
4. After SA is complete; use SD to ensure the solution is a local optimum.
5. Solve multiple examples to test the performance of the heuristics (see Illustrative Examples).

Chapter 4: Methodology

SA Algorithm Pseudo-Code

1. Obtain initial π solution S where S is the solution space (set of all feasible solutions).
2. Select initial temperatre T(0), T > 0.
3. Set temperature counter t=0, iteration counter at current temperature n=0, and total iteration counter k=0.
4. Generate state π', a neighbor of π using local search technique.
5. Obtain ΔTC = TC(π) – TC(π').
6. If ΔTC >0, then π = π' (minimization). If ΔTC<0 and exp(ΔTC/T)>rand(0,1), then π=π'.
    1. Else keep π.
    2. Set k = k+1 and n=n+1.
    3. Repeat steps 4-6 until n = N(t) (epoch length) or stopping criterion has been reached (e.g., k = max_itera).
7. Set t=t+1, T=T(t), n=0. Go step 4.

Makespan Pseudo Code

1. Let
    a. t(i,j,m) = processing time, at stage i, on machine m of job in position j in processing sequence P
    b. T(i,j,m) = completion time, at stage i, on machine m of job in position j in processing sequence P
    c. TM(i,m)=current processing time of machine m at stage i.
2. The establishment of the makespan table is as follows:
    a. Let I = number of stages
    b. Let N = number of jobs to be scheduled
    c. Let M = number of machines
3. Develop I tables with N internal rows and M internal columns. Add 2 columns to the the table to depict the job-processing sequence under consideration and a top row to head the columns.
4. For each row associated with each job in position j, enter the process times of that job in the upper half of the cells for that row (i.e., enter t(i,j,m) for each j).
5. For table i=1 j,m=1..M: T(1,j,m)=t(1,j,m)=TM(1,m)
6. For table i=1 and j>M: T(1,j,m) = min{TM(1,m)}+t(1,j,m)=TM(1,m)

7. After table 1 is complete, organize products in ascending order by T(1,j,m). This is the new processing sequence P.
8. For table i=2, j,m=1..M: T(i,j,m)=T(i-1=1=,j,m)+t(i=2,j,m)+TM(i=2,m)
9. For table i=2 and j>M: T(i=2,j,m) = max{T(i-1=1,j,m),min{TM(i=2,m)}}+t(i=2,j,m)=TM(i=2,m)
10. After table i=2 is complete, organize products in ascending order by T(i=2,j,m). This is the new processing sequence P.
11. Repeat steps 8-10 for Tables 3 through I.
12. The max of T(I,j,m) or TM(I,m) will give the Total Makespan of the whole process.

Steepest Descent Pseudo Code

1. Select a starting solution s_0 in S (a set of feasible solutions)
2. Select s in $N(s_0)$ such that $f(s) < f(s_0)$ by steepest descent (largest improvement/decrease)
3. Replace s_0 by s.
4. Repeat steps 2-3 until $f(s) \geq f(s_0)$ for all s in $N(s_0)$.

Chapter 5: Computational Results

Illustrative Example 1:

Consider a 7 job, 4 stage (1 machine per stage) FS problem with processing time as shown in table 1. Problem taken from Kumar et al (2014).

Table I: Illustrative Example 1

JOBS	M_1	M_2	M_3	M_4
J_1	3	1	4	12
J_2	8	0	5	15
J_3	11	3	8	10
J_4	4	7	3	8
J_5	5	5	1	10
J_6	10	2	0	13
J_7	2	5	6	9

In the Gupta heuristic algorithm all the jobs are divided into two groups by comparing the dispensation times of the first machine and the last machine in each job. For every group, calculate the sum of processing times of any two adjacent tasks in a job and find the minimum processing time, and then schedules the jobs in sorting order according to their minimum summed processing times (Kumar et al., 2014).

Table II: Makespan Calculation using GUPTA's Algorithm for Illustrative Example 1

JOBS	M_1	M_2	M_3	M_4
J_1	0-3	3-4	4-8	8-20
J_2	3-11	11-11	11-16	20-35
J_5	11-16	16-21	21-22	35-45
J_7	16-18	21-26	26-32	45-54
J_4	18-22	26-33	33-36	54-62
J_3	22-33	33-36	36-44	62-72
J_6	33-44	44-46	46-46	72-85

The best sequence using GUPTA's Algorithm is = {J1, J2, J5, J7, J4, J3, J6} , which gives a makespan of 85.

This problem was also solved using a novel hybrid permutation flow shop scheduling developed by Kumar et al. (2014). The satisfaction criteria to use this permutation is as follows: If the maximum processing time on Machine 1 is greater than or equal to the minimum processing time on machine m_1, m_2,… m_{M-1} and if the minimum processing time on machine m_M is greater than or equal to the maximum processing time on machine m_2, m_3,…m_{M-1}. The conditions for the above problem are met; therefore, two hypothetical machines X and Y are introduced, respectively. X is the sum of the processing times of the first three machines, while Y is the sum of the last three machines.

Table III: Hypothetical Machines for Illustrative Example 1

JOBS	X	Y
J_1	8	17
J_2	13	20
J_3	22	21
J_4	14	18
J_5	11	16
J_6	12	15
J_7	13	20

Ordered Group in Ascending X= {J1, J5, J6, J7, J2, J4} due to Y > X for these respective machines.
Ordered Group in Descending Y= {J3} due to X > Y for this machine.

This concludes in this schedule with the below makespan table: {J1, J5, J6, J7, J2, J4, J3}

Table IV: Makespan Using a Novel Hybrid Permutation Flow Shop Scheduling for Illustrative Example 1				
JOBS	**M_1**	**M_2**	**M_3**	**M_4**
J_1	0-3	3-4	4-8	8-20
J_5	3-7	7-14	14-17	20-30
J_6	7-17	17-19	19-19	30-43
J_7	17-19	19-24	24-30	43-52
J_2	19-27	27-27	30-35	52-67
J_4	27-31	31-38	38-41	67-75
J_3	31-42	42-45	45-53	75-85

According to Kumar et al. (2014) the makespan for this generated sequence is 85. This is after correcting the processing time for Job 5 on Machine 4 from 8 to 10 (the makespan in the literature is incorrectly calculated to 83).

The initial parameters and the solution summary can be seen below.

Initial Parameters:

- The initial temperature is: 15
- The epoch length is: 70 (10 × N)
- The maximum iterations without improvement : 350 (50 * N)
- The cooling parameter is: 0.9

Solution Summary:

- Best Found Solution
- The best schedule is: 1 2 5 3 4 6 7
- The time for the best schedule is: 85 time units
- Best Solution found at iteration #5
- The iterations at the current temperature: 7
- The total iterations is: 357
- The number of temperature changes: 5
- The time taken to solve this problem is 0.21387 seconds

See makespan Table V below for best job sequence and job completion times for stages 1, 2, 3, and 4 using SA heuristic. An attempt to improve the sequence was made by applying steepest descent heuristic to the solution. See neighborhood of solutions in Table VI below. It is important to note that the schedule did not improve.

Table V: Makespan Table for Illustrative Example 1

Job	Stage 1	Stage 2	Stage 3	Stage 4
1	3	4	8	20
2	11	11	16	35
5	16	21	22	45
3	27	30	38	55
4	31	38	41	63
6	41	43	43	76
7	43	48	54	85

Table VI: Steepest Descent Neighborhood of Solutions and Makespans for Solution 1 2 5 3 4 6 7 of Illustrative Example 1

2	1	5	3	4	6	7	90	1	2	3	5	4	6	7	85
5	2	1	3	4	6	7	88	1	2	4	3	5	6	7	85
3	2	5	1	4	6	7	99	1	2	6	3	4	5	7	85
4	2	5	3	1	6	7	91	1	2	7	3	4	6	5	85
6	2	5	3	4	1	7	89	1	2	5	4	3	6	7	85
7	2	5	3	4	6	1	90	1	2	5	6	4	3	7	85
1	5	2	3	4	6	7	85	1	2	5	7	4	6	3	85
1	3	5	2	4	6	7	90	1	2	5	3	6	4	7	85
1	4	5	3	2	6	7	85	1	2	5	3	7	6	4	85
1	6	5	3	4	2	7	85	1	2	5	3	4	7	6	85
1	7	5	3	4	6	2	85								

Illustrative Example 2:

Consider ten jobs and four stages, with each stage having 4 parallel, uniform machines (Table VII) (Tyagi, 2016).

The Construction Algortihm used (Tyagi, 2016) utilizes the Minimum Processing Time Selective Approach (MPTSA) and the Longest Processing Times (LPT) approach. The problem first categorizes the four-stage (four machines per stage) flexible flowshop problem into four four-stage, single machine flexible flowshop scheduling problems (seen in the equations below). Jobs are then assigned to these four separate problems using MPTSA (Tyagi, 2016). LPT is then used in each individual flowshop to schedule the respective jobs. The makespan of 34 is calculated using the maximum of each of the four flowshops.

$$F_{S_1} = M_{11} + M_{21} + M_{31} + M_{41}\ for\ assigned\ jobs: j_7, j_8$$
$$F_{S_2} = M_{12} + M_{22} + M_{32} + M_{42}\ for\ assigned\ jobs: j_1, j_3, j_{10}$$
$$F_{S_3} = M_{13} + M_{23} + M_{33} + M_{43}\ for\ assigned\ jobs: j_4, j_6, j_9$$
$$F_{S_4} = M_{14} + M_{24} + M_{34} + M_{44}\ for\ assigned\ jobs\ j_2, j_5$$

Table VII: Ten Job, Four Stage FF Data for Illustrative Example 2

	Stages			
	S_1	**S_2**	**S_3**	**S_4**
Jobs	**M_{1i}**	**M_{2i}**	**M_{3i}**	**M_{4i}**
J_1	6	2	9	5
J_2	8	3	4	2
J_3	5	4	7	8
J_4	6	5	2	4
J_5	5	2	4	1
J_6	3	4	1	2
J_7	1	3	5	2
J_8	2	7	4	5
J_9	8	4	3	6
J_{10}	2	1	6	3

This example is also solved using the Heuristic Algorithm Using Palmer Approach developed by Tyagi. Jobs are assigned to four separate flowshop problems as per the method above. However, now a slope for every job is calculated, and the jobs are sequenced using Longest Slope Value (LSV) (Tyagi, 2016) in descending order. The slope Y_i for n^{th} jobs (i=1 to n) for every category of the flow shop scheduling (F_{S_q}) on each machine center stage (S_q) is as follows:

$Y_i = -\sum_{k=1}^{q}\{q-(2k-1)]m_{i(F_{S_q})}$ (Calculated slopes can be seen in Table VIII below)

Table VIII: Calculated Slope Value for Palmer Based Heuristic Algorithm for Illustrative Example 2						
	Jobs j_i	m_{1i} **k=1**	m_{2i} **k=2**	m_{3i} **k=3**	m_{4i} **k=4**	Y_i
F_{S_1}	j_7	3	3	-5	-6	5
	j_8	6	7	-4	-9	6
F_{S_2}	j_1	18	2	-9	-15	4
	j_3	15	4	-7	-24	12
	j_{10}	6	1	-6	-9	8
F_{S_3}	j_4	18	5	-3	-12	-9
	j_6	9	4	-2	-6	-6
	j_9	24	4	-3	-18	-7
F_{S_4}	j_2	24	3	-4	-6	-17
	j_5	15	2	-4	-3	-10

The makespan of 36 is calculated using the maximum of each of the flowshops. Sequencing, solutions, and comparisons for both methods can be seen below in Table IX, Table X, Table XI, and Table XII.

Table IX: Sequencing of Jobs According to LPT and LSV for Illustrative Example 2				
Machine Center Stage K	**Flowshop Categories (F_{S_k})**	**Assigned Jobs**	**Sequenced Jobs for LPT**	**Sequenced Jobs for LSV**
1	F_{S_1}	j_7, j_8	$j_7 > j_8$	$j_7 > j_8$
2	F_{S_2}	j_1, j_3, j_{10}	$j_3 > j_1 > j_{10}$	$j_3 > j_{10} > j_1$
3	F_{S_3}	j_4, j_6, j_9	$j_9 > j_4 > j_6$	$j_6 > j_9 > j_4$
4	F_{S_4}	j_2, j_5	$j_2 > j_5$	$j_5 > j_2$

Table X: In-Out Table for FFS using Constructive Heuristic Algorithm for Illustrative Example 2						
	Jobs j_i	S_1 **In---Out**	S_2 **In---Out**	S_3 **In---Out**	S_4 **In---Out**	
F_{S_1}	j_8	0---2	2---6	6---12	12---15	
	j_7	2---3	6---12	12---17	17---19	$C_{max_{F_{S_1}}}$
F_{S_2}	j_3	0---5	5---9	9---16	16---24	
	j_1	5---11	11---13	16---25	25---30	
	j_{10}	11---13	13---14	25---31	31---34	$C_{max_{F_{S_2}}}$
F_{S_3}	j_9	0---8	8---12	12---15	15---21	
	j_4	8---14	14---19	19---21	21---25	
	j_6	14---17	19---23	23---24	25---27	$C_{max_{F_{S_3}}}$
F_{S_4}	j_2	0---8	8---11	11---15	15---17	
	j_5	8---13	13---15	15---19	19---20	$C_{max_{F_{S_4}}}$

Table XI: In-Out Table for FFS using Palmer Based Heuristic Algorithm for Illustrative Example 2						
	Jobs	S_1	S_2	S_3	S_4	
	j_i	**In---Out**	**In---Out**	**In---Out**	**In---Out**	
F_{S_1}	j_8	0---2	2---6	6---12	12---15	
	j_7	2---3	6---12	12---17	17---19	$C_{max_{F_{S_1}}}$
F_{S_2}	j_3	0---5	5---9	9---16	16---24	
	j_{10}	5---17	9---10	16---22	24---27	
	j_1	7---13	13---15	22---31	31---36	$C_{max_{F_{S_2}}}$
F_{S_3}	j_6	0---3	3---7	7---8	8---10	
	j_9	3---11	11---15	15---18	18---24	
	j_4	11---17	17---22	22---24	24---28	$C_{max_{F_{S_3}}}$
F_{S_4}	j_5	0---5	5---7	7---11	11---12	
	j_2	5---13	13---16	16---20	20---22	$C_{max_{F_{S_4}}}$

Table XII: Comparative Study Between Constructive and Palmer Based Heuristic Algorithms for Illustrative Example 2		
Makespan	**Constructive**	**Palmer**
$C_{max_{F_{S_1}}}$	19	19
$C_{max_{F_{S_2}}}$	34	36
$C_{max_{F_{S_3}}}$	27	28
$C_{max_{F_{S_4}}}$	20	22
$C_{max_{FLEX}}$	100	105

The initial parameters and the solution summary can be seen below.

Initial Parameters:

- The initial temperature is: 15
- The epoch length is: 100 (50 * N)
- The maximum iterations without improvement : 500 (50 * N)
- The cooling parameter is: 0.9

Solution Summary:

- Best Found Solution
- The best schedule is: 3 8 10 4 7 1 9 2 6 5
- The time for the best schedule is: 24 time units
- Best Solution found at iteration #13

- The iterations at the current temperature: 15
- The total iterations is: 515
- The number of temperature changes: 5
- The time taken to solve this problem is 0.28794 seconds

See makespan Tables XIII, XIV, XV, and XVI below for job and machine assignments (upper left corner) and completion times on machines (lower right corner) for stages 1, 2, 3, and 4, respectively. The solution (24 time units) can be seen in Table XVI, which is the maximum of all the machine times in the final stage. An attempt to improve the sequence was made by applying the steepest descent heuristic to the solution. See neighborhood of solutions in Table XVII below. Again, the schedule did not improve.

Table XIII: Makespan Table for Illustrative Example 2, Stage 1

Machine 1	Machine 2	Machine 3	Machine 4
3 / 5	8 / 2	10 / 2	4 / 6
2 / 13	7 / 3	1 / 8	6 / 9
NA / NA	9 / 11	5 / 13	NA / NA

Table XIV: Makespan Table for Illustrative Example 2, Stage 2

Machine 1	Machine 2	Machine 3	Machine 4
8 / 9	10 / 3	7 / 6	3 / 9
6 / 13	4 / 11	1 / 10	9 / 15
NA / NA	5 / 15	2 / 16	NA / NA

Table XV: Makespan Table for Illustrative Example 2, Stage 3			
Machine 1	**Machine 2**	**Machine 3**	**Machine 4**
10 9	7 11	8 13	3 16
1 19	4 13	9 18	2 20
NA NA	6 14	NA NA	NA NA
NA NA	5 19	NA NA	NA NA

Table XVI: Makespan Table for Illustrative Example 2, Stage 4			
Machine 1	**Machine 2**	**Machine 3**	**Machine 4**
10 12	7 13	8 18	4 17
6 16	3 24	5 20	1 24
9 24	NA NA	2 22	NA NA

Table XVII: Steepest Descent Neighborhood of Solutions and Makespans for Solution 3 8 10 4 7 1 9 2 6 5 of Illustrative Example 2																					
8	3	10	4	7	1	9	2	6	5	24	3	8	5	4	7	1	9	2	6	10	25
10	8	3	4	7	1	9	2	6	5	24	3	8	10	7	4	1	9	2	6	5	24
4	8	10	3	7	1	9	2	6	5	24	3	8	10	1	7	4	9	2	6	5	25
7	8	10	4	3	1	9	2	6	5	25	3	8	10	9	7	1	4	2	6	5	25
1	8	10	4	7	3	9	2	6	5	26	3	8	10	2	7	1	9	4	6	5	24
9	8	10	4	7	1	3	2	6	5	28	3	8	10	6	7	1	9	2	4	5	25
2	8	10	4	7	1	9	3	6	5	30	3	8	10	5	7	1	9	2	6	4	25
6	8	10	4	7	1	9	2	3	5	30	3	8	10	4	1	7	9	2	6	5	24
5	8	10	4	7	1	9	2	6	3	32	3	8	10	4	9	1	7	2	6	5	27
3	10	8	4	7	1	9	2	6	5	24	3	8	10	4	2	1	9	7	6	5	27
3	4	10	8	7	1	9	2	6	5	24	3	8	10	4	6	1	9	2	7	5	27
3	7	10	4	8	1	9	2	6	5	24	3	8	10	4	5	1	9	2	6	7	26
3	1	10	4	7	8	9	2	6	5	27	3	8	10	4	7	9	1	2	6	5	25
3	9	10	4	7	1	8	2	6	5	25	3	8	10	4	7	2	9	1	6	5	27
3	2	10	4	7	1	9	8	6	5	27	3	8	10	4	7	6	9	2	1	5	28
3	6	10	4	7	1	9	2	8	5	28	3	8	10	4	7	5	9	2	6	1	28
3	5	10	4	7	1	9	2	6	8	27	3	8	10	4	7	1	2	9	6	5	26
3	8	4	10	7	1	9	2	6	5	24	3	8	10	4	7	1	6	2	9	5	27
3	8	7	4	10	1	9	2	6	5	24	3	8	10	4	7	1	5	2	6	9	27
3	8	1	4	7	10	9	2	6	5	26	3	8	10	4	7	1	9	6	2	5	24
3	8	9	4	7	1	10	2	6	5	27	3	8	10	4	7	1	9	5	6	2	25
3	8	2	4	7	1	9	10	6	5	27	3	8	10	4	7	1	9	2	5	6	24
3	8	6	4	7	1	9	2	10	5	27											

Chapter 6: Conclusion/Future Work

In this study, SA solved the illustrative examples using a very small amount of computation time (<0.3 seconds). A data error was found in the example problem of Tyagi (2016) which was corrected so the solutions could be compared. SD did not improve upon the solutions found by SA. This was most likely due to the small size of the problems.

In future work, assumptions not deemed realistic (see Problem Statement) will be relaxed. These realistic conditions will add complexity to the problem. SA can also be compared to Tabu Search or Genetic Algorithms, for comparison and/or validation of solution. It would also be very beneficial to find large-scale, non-randomized data (with good or optimal solutions provided) to test the limits of the proposed SA heuristic and other solution methods.

References

Arthanari, T., and Ramamurthy, K. (1971). An Extension of Two Machines Sequencing Problem. *Opsearch,* 8, 10-22.

Blazewicz, J., M. Dror, G. Pawlak, K. Stecke (1992), “Scheduling parts through a two-stage tandem flexible flow shop”, Working paper 699, Division of research, school of Business Administration, The University of Michigan.

Brah, S. and Hunsucker, J. (1991). Branch and bound algorithm for the flowshop with multiple processors. *European Journal of Operational Research,* 51, 88-99.

Brockmann, K., W. Dangelmaier, N. Holthöfer (1997),”Parallel Branch & Bound Algorithm for makespan Optimal Scheduling in flowshops with multiple processors”, Operations Research Proceedings 1997, Selected Papers of the Symposium on Operations Research (SOR 97), 428-433.

Chen, Bo (1995). Analysis of Classes of Heuristics for Scheduling a Two-Stage Flowshop with Parallel Machines at One Stage. *Journal of the Operational Research Society*, 46, 234-244.

Crowder, Bret. (2006). Minimizing the Makespan in a Flexible Flowshop with Sequence Dependent Setup Times, Uniform Machines, and Limited Buffers. Thesis submitted to the College of Engineering and Mineral Resources.

Gilmore, P., and Gomory, R. (1964). Sequencing a One State-Variable Machine a Solvable Case of the Traveling Salesman Problem. *Operations Research,* 12, 655-679.

Guðmundsdóttir, Rannveig. (2012). Production Scheduling in a Campaign Based Flexible Flow Shop. Research thesis submitted to the School of Science and Engineering at Reykjavík University.

Gupta, J.N.D. (1988). Two-Stage, Hybrid Flowshop Scheduling Problem*. Operational Research Society,* 39(4), 359-364.

Johnson, S.M. (1954). Optimal Two-and Three-Stage Production Schedules with Setup Times included. *Naval Research Logistics Quarterly*, 1(1), 61-67.

Koulamas, C., and Kyparisis, G. (2000). Asymptotically Optimal Linear Time Algorithms for Two-

Stage and Three-Stage Flexible Flowshops. Naval Research Logistics, Volume47, Issue3, April 2000, 259-268.

Metropolis, N., A. Rosenbluth, M. Rosenbluth, A. Teller (1953), "Equation of State Calculations by Fast Computing Machines", J. Chem. Phys., 21, 6, 1087-1092

Portman, M., Vignier, A., Dardilhac, D., and Dezalay, D. (1998). Branch and bound crossed with GA to solve hybrid flowshops. *European Journal of Operational Research*, 107, 389-400.

Salvador, M. S. (1973), “A Solution of a Special Class of Flow Shop Scheduling Problems”, In proceedings of the symposium on the theory of scheduling and its Applications, pp. 83-91, Springer-Verlag, Berlin.

Sawik, T. (2000). Mixed Integer Programming for Scheduling Flexible Flow Lines with Limited Intermediate Buffers. *Mathematical and Computer Modelling*, 31, 39-52.

Sawik, T. (2002). An Exact Approach for Batch Scheduling in Flexible Flow Lines with Limited Intermediate Buffers. Mathematical and Computer Modelling. Volume 36, Issues 4–5, 461-471

Sethanan, Kanchana. (2001). Scheduling Flexible Flowshops with Sequence Dependent Setup Times. Dissertation submitted to the College of Engineering and Mineral Resources at West Virginia University.

Shieh, Alireza. (2004). A Simulated Annealing Approach for Flexible Flowshop Scheduling to Maximize Flexibility. Thesis submitted to the College of Engineering and Mineral Resources at West Virginia University.

Soewandi, H., and Elmaghraby, S. (2001). Sequencing three-stage flexible flowshops with identical machines to minimize makespan. *IIE Transactions*, 33, 985-993.

Tyagi, Neelam, Tripathi, R.P., and Chandramoul, A.B.. (2016). Flexible Flowshop Scheduling Model with Four Stages. Indian Journal of Science and Technology, Vol 9(42).

Verma, S., and Dessouky, M. (1999). Multistage Hybrid Flowshop Scheduling With Identical Jobs and Uniform Parallel Machines. *Journal of Scheduling*, 2, 135-150.

Wittrock, R. (1988). An Adaptable Scheduling Algorithm for Flexible Flow Lines. *Operations Research Society of America*, 36(3), 445-453

Appendix

%%
%%%%%%%%%%%%%%%%%%%%%%%%
%Problem Report - Pharmaceutical Scheduling using Simulated Annealing
%Illustrative Example 2
%%
%%%%%%%%%%%%%%%%%%%%%%%%%

%% Input Data

%Processing Time Table
ProcessTime=[6 2 9 5;
        8 3 4 2;
        5 4 7 8;
        6 5 2 4;
        5 2 4 1;
        3 4 1 2;
        1 3 5 2;
        2 7 4 5;
        8 4 3 6;
        2 1 6 3];

M=4; %Number of machines @ each stage
N=10; %Total number of products
I=4; %Number of stages

P=zeros(2,N); %Creates empty processing sequence table (row1=jobs, row2=completion time after each stage)
P(1,:)=randperm(N); %Randomly chooses initial sequence (this will be handled with the SA portion of the code)
s0=P(1,:);

CompletionTime=zeros(N,M); %Creates empty Makespan table
MachineCT=zeros(1,M); %Table keeps track of completion time for each machine

```
MachinePN=ones(1,M); %Table keeps track of number of products that have been on each
machine
Schedule=zeros(N,M); %Overall schedule of all machines/all stages

%% Obtain Initial Solution
%Stage 1

i=1; %current stage

for j = 1:M %For the first product on each machine
   CompletionTime(1,j)=ProcessTime(P(1,j),i); %Completion time is the same as processing
time
   Schedule(1,j)=P(1,j); %Keeps track of products on machines
   P(2,j)=CompletionTime(1,j); %Updates current completion time for each product during
Stage 1
   MachineCT(j)=CompletionTime(1,j); %Updates current completion time for each machine
end

for j=M+1:N %For the remaining products during stage 1
   [minMCT,machine]=min(MachineCT); %Finds which machine gets the next product
   MachinePN(machine)=MachinePN(machine)+1; %Updates the number of products on each
machine
   CompletionTime(MachinePN(machine),machine)=minMCT+ProcessTime(P(1,j),i);
%Updates the completion time of the chosen machine
   Schedule(MachinePN(machine),machine)=P(1,j); %Keeps track of products on machines
   P(2,j)=CompletionTime(MachinePN(machine),machine); %Updates current completion time
for each product for Stage 1
   MachineCT(machine)=CompletionTime(MachinePN(machine),machine); %Updates current
completion time for each machine
end

CompletionTime( ~any(CompletionTime,2), : ) = [];  %deletes empty rows
Schedule( ~any(Schedule,2), : ) = [];  %deletes empty rows
```

```
%Reorders products in ascending order based on completion time
[Y,sortedindex]=sort(P(2,:));
P=P(:,sortedindex);

%Stages 2 through I

for i=2:I %For all stages beyond Stage 1
   CompletionTime=zeros(N,M); %Erases makespan table
   MachineCT=zeros(1,M); %Erases machine completion time table
   MachinePN=ones(1,M); %Erases machine product number table
   Schedule=zeros(N,M); %Ereases overall schedule

   for j = 1:M %For the first product on each machine
   CompletionTime(1,j)=ProcessTime(P(1,j),i)+P(2,j); %The product's completion time from the
previous stage plus processing time
   Schedule(1,j)=P(1,j); %Keeps track of products on machines
   P(2,j)=CompletionTime(1,j); %Updates current completion time for each product for Stage i
   MachineCT(j)=CompletionTime(1,j); %Updates current completion time for each machine
   end

   for j=M+1:N %For the remaining products
   [minMCT,machine]=min(MachineCT); %Finds which machine gets the next product
   MachinePN(machine)=MachinePN(machine)+1; %Updates the number of products on each
machine
   maxTM=max(minMCT,P(2,j)); %Completion time is the maximum of current completion time
of the selected machine or the product's current completion time from the previous stage
   CompletionTime(MachinePN(machine),machine)=maxTM+ProcessTime(P(1,j),i); %Updates
completion time of chosen machine
   Schedule(MachinePN(machine),machine)=P(1,j); %Keeps track of products on machines
   P(2,j)=CompletionTime(MachinePN(machine),machine); %Updates current completion time
for each product for Stage i
   MachineCT(machine)=CompletionTime(MachinePN(machine),machine); %Updates current
completion time for each machine

   end
```

```
    CompletionTime( ~any(CompletionTime,2), : ) = [];  %deletes empty rows
    Schedule( ~any(Schedule,2), : ) = [];  %deletes empty rows



    %Reorders products in ascending order based on completion time
    [Y,sortedindex]=sort(P(2,:));
    P=P(:,sortedindex);

end

%The makespan is the maximum completion time at the end of the last stage
MakeSpan=max(MachineCT);
fs0=MakeSpan;

sbest=s0; %Sets initial as best
zbest=fs0;
ibest=0;



%% Initial Temp and Counters

T=15; %15; %Initial/Current Temperature (will change)
Tinit=T; %Initial Temperature for summary at the end
t=0; %temp counter
n=0; %iteration counter at current temp
kprime=0; %total # of iterations counter
kimprove=0; %iterations without improvment counter
Nt=10*N; %epoch length
max_itera=100*N; %max iterations without improvment
alpha=.9; %cooling schedule
```

```
disp(['The initial temperature is: ' num2str(T)]);
disp(['The epoch length is: ' num2str(Nt)]);
disp(['The maximum iterations without improvement : ' num2str(max_itera)]);
disp(['The cooling parameter is: ' num2str(alpha)]);
disp(' ');

%% Randomly Generate a Neighbor
tic;
while kimprove <= max_itera %stopping criteria: iterations without improvment

u=randi([1 N],1); %random numbers for pairwise exchange

v=randi([1 N],1);

while v==u %prevents u from equaling v
   v=randi([1 N],1);
end

s=s0; %sets s as s0 to perform exchange

temp=s(u);
s(u)=s(v);
s(v)=temp;


%% Determine Change in OFV

P=zeros(2,N); %Creates empty processing sequence table (row1=jobs, row2=completion time after each stage)
P(1,:)=s; %Randomly chooses initial sequence (this will be handled with the SA portion of the code)

CompletionTime=zeros(N,M); %Creates empty Makespan table
MachineCT=zeros(1,M); %Table keeps track of completion time for each machine
```

```
MachinePN=ones(1,M); %Table keeps track of number of products that have been on each
machine
Schedule=zeros(N,M); %Overall schedule of all machines/all stages

%Stage 1

i=1; %current stage

for j = 1:M %For the first product on each machine
   CompletionTime(1,j)=ProcessTime(P(1,j),i); %Completion time is the same as processing
time
   Schedule(1,j)=P(1,j); %Keeps track of products on machines
   P(2,j)=CompletionTime(1,j); %Updates current completion time for each product during
Stage 1
   MachineCT(j)=CompletionTime(1,j); %Updates current completion time for each machine
end

for j=M+1:N %For the remaining products during stage 1
   [minMCT,machine]=min(MachineCT); %Finds which machine gets the next product
   MachinePN(machine)=MachinePN(machine)+1; %Updates the number of products on each
machine
   CompletionTime(MachinePN(machine),machine)=minMCT+ProcessTime(P(1,j),i);
%Updates the completion time of the chosen machine
   Schedule(MachinePN(machine),machine)=P(1,j); %Keeps track of products on machines
   P(2,j)=CompletionTime(MachinePN(machine),machine); %Updates current completion time
for each product for Stage 1
   MachineCT(machine)=CompletionTime(MachinePN(machine),machine); %Updates current
completion time for each machine
end

CompletionTime( ~any(CompletionTime,2), : ) = [];  %deletes empty rows
Schedule( ~any(Schedule,2), : ) = [];  %deletes empty rows


%Reorders products in ascending order based on completion time
```

```
[Y,sortedindex]=sort(P(2,:));
P=P(:,sortedindex);

%Stages 2 through I

for i=2:I %For all stages beyond Stage 1
   CompletionTime=zeros(N,M); %Erases makespan table
   MachineCT=zeros(1,M); %Erases machine completion time table
   MachinePN=ones(1,M); %Erases machine product number table
   Schedule=zeros(N,M); %Ereases overall schedule

   for j = 1:M %For the first product on each machine
   CompletionTime(1,j)=ProcessTime(P(1,j),i)+P(2,j); %The product's completion time from the
previous stage plus processing time
   Schedule(1,j)=P(1,j); %Keeps track of products on machines
   P(2,j)=CompletionTime(1,j); %Updates current completion time for each product for Stage i
   MachineCT(j)=CompletionTime(1,j); %Updates current completion time for each machine
   end

   for j=M+1:N %For the remaining products
   [minMCT,machine]=min(MachineCT); %Finds which machine gets the next product
   MachinePN(machine)=MachinePN(machine)+1; %Updates the number of products on each
machine
   maxTM=max(minMCT,P(2,j)); %Completion time is the maximum of current completion time
of the selected machine or the product's current completion time from the previous stage
   CompletionTime(MachinePN(machine),machine)=maxTM+ProcessTime(P(1,j),i); %Updates
completion time of chosen machine
   Schedule(MachinePN(machine),machine)=P(1,j); %Keeps track of products on machines
   P(2,j)=CompletionTime(MachinePN(machine),machine); %Updates current completion time
for each product for Stage i
   MachineCT(machine)=CompletionTime(MachinePN(machine),machine); %Updates current
completion time for each machine

   end
```

```
    CompletionTime( ~any(CompletionTime,2), : ) = [];  %deletes empty rows
    Schedule( ~any(Schedule,2), : ) = [];  %deletes empty rows



    %Reorders products in ascending order based on completion time
    [Y,sortedindex]=sort(P(2,:));
    P=P(:,sortedindex);

end

%The makespan is the maximum completion time at the end of the last stage
MakeSpan=max(MachineCT);
fs=MakeSpan;



deltaf=fs0-fs; %difference between OFV of current and neighbor solution
nonimprove=exp(deltaf/T); %value used for nonimproving solution


if deltaf > 0 %choose improving solution
    s0=s;
    fs0=fs;
elseif deltaf < 0 & nonimprove > rand(0,1) %choose non-improving solution
    s0=s;
    fs0=fs;
end

if fs0 < zbest %new best solution
    sbest=s0;
    zbest=fs0;
    ibest=kprime; %best solution found at iteration #
    kimprove=0;
```

```
else
   kimprove=kimprove+1; %no improvment
end



n=n+1; %increase iteration @ current temp
kprime=kprime+1; %increase total iteration

if n==Nt
   T=T*alpha; %change current temp
   n=0; %reset iteration @ current temp counter
   t=t+1; %increase temperature change counter
end




end


%final solution
disp('Summary');
disp(' ');
disp('Best Found Solution');
disp(['The best schedule is: ' num2str(sbest)]);
disp(' ');

%Recalculating Optimal Solution
P=zeros(2,N); %Creates empty processing sequence table (row1=jobs, row2=completion time
after each stage)
P(1,:)=sbest; %Randomly chooses initial sequence (this will be handled with the SA portion of
the code)

CompletionTime=zeros(N,M); %Creates empty Makespan table
```

```
MachineCT=zeros(1,M); %Table keeps track of completion time for each machine
MachinePN=ones(1,M); %Table keeps track of number of products that have been on each
machine
Schedule=zeros(N,M); %Overall schedule of all machines/all stages

%Stage 1

i=1; %current stage

for j = 1:M %For the first product on each machine
   CompletionTime(1,j)=ProcessTime(P(1,j),i); %Completion time is the same as processing
time
   Schedule(1,j)=P(1,j); %Keeps track of products on machines
   P(2,j)=CompletionTime(1,j); %Updates current completion time for each product during
Stage 1
   MachineCT(j)=CompletionTime(1,j); %Updates current completion time for each machine
end

for j=M+1:N %For the remaining products during stage 1
   [minMCT,machine]=min(MachineCT); %Finds which machine gets the next product
   MachinePN(machine)=MachinePN(machine)+1; %Updates the number of products on each
machine
   CompletionTime(MachinePN(machine),machine)=minMCT+ProcessTime(P(1,j),i);
%Updates the completion time of the chosen machine
   Schedule(MachinePN(machine),machine)=P(1,j); %Keeps track of products on machines
   P(2,j)=CompletionTime(MachinePN(machine),machine); %Updates current completion time
for each product for Stage 1
   MachineCT(machine)=CompletionTime(MachinePN(machine),machine); %Updates current
completion time for each machine
end

CompletionTime( ~any(CompletionTime,2), : ) = [];  %deletes empty rows
Schedule( ~any(Schedule,2), : ) = [];  %deletes empty rows

display(['Stage ' num2str(i)]) %Displays Stage# in command window
```

```
display(Schedule) %Displays machine/product schedule for Stage 1
display(CompletionTime) %Displays completion times for Stage 1

%Reorders products in ascending order based on completion time
[Y,sortedindex]=sort(P(2,:));
P=P(:,sortedindex);

%Stages 2 through I

for i=2:I %For all stages beyond Stage 1
   CompletionTime=zeros(N,M); %Erases makespan table
   MachineCT=zeros(1,M); %Erases machine completion time table
   MachinePN=ones(1,M); %Erases machine product number table
   Schedule=zeros(N,M); %Ereases overall schedule

   for j = 1:M %For the first product on each machine
   CompletionTime(1,j)=ProcessTime(P(1,j),i)+P(2,j); %The product's completion time from the
previous stage plus processing time
   Schedule(1,j)=P(1,j); %Keeps track of products on machines
   P(2,j)=CompletionTime(1,j); %Updates current completion time for each product for Stage i
   MachineCT(j)=CompletionTime(1,j); %Updates current completion time for each machine
   end

   for j=M+1:N %For the remaining products
   [minMCT,machine]=min(MachineCT); %Finds which machine gets the next product
   MachinePN(machine)=MachinePN(machine)+1; %Updates the number of products on each
machine
   maxTM=max(minMCT,P(2,j)); %Completion time is the maximum of current completion time
of the selected machine or the product's current completion time from the previous stage
   CompletionTime(MachinePN(machine),machine)=maxTM+ProcessTime(P(1,j),i); %Updates
completion time of chosen machine
   Schedule(MachinePN(machine),machine)=P(1,j); %Keeps track of products on machines
   P(2,j)=CompletionTime(MachinePN(machine),machine); %Updates current completion time
for each product for Stage i
```

```
    MachineCT(machine)=CompletionTime(MachinePN(machine),machine); %Updates current
completion time for each machine

    end

    CompletionTime( ~any(CompletionTime,2), : ) = [];  %deletes empty rows
    Schedule( ~any(Schedule,2), : ) = [];  %deletes empty rows

    display(['Stage ' num2str(i)]) %Displays Stage# in command window
    display(Schedule) %Displays machine/product schedule for Stage i
    display(CompletionTime) %Displays completion times for Stage i


    %Reorders products in ascending order based on completion time
    [Y,sortedindex]=sort(P(2,:));
    P=P(:,sortedindex);

end



disp(['The time for the best schedule is: ' num2str(zbest) ' time units']);
disp(['Best Solution found at iteration #' num2str(ibest)]);
disp(' ');
disp('Iteration Information');
disp(['The iterations at the current temperature: ' num2str(n)]);
disp(['The total iterations is: ' num2str(kprime)]);
disp(['The number of temperature changes: ' num2str(t)]);
disp(['The time taken to solve this problem is ' num2str(toc) ' seconds']);
disp(' ');


%Steepest Descent

Continue =1;
```

```
NumOfSolns=N*(N-1)/2;
NbhdOfSolns=zeros(NumOfSolns,N);
Time=zeros(NumOfSolns,1);

while Continue == 1

x=0;
P=zeros(2,N); %Creates empty processing sequence table (row1=jobs, row2=completion time after each stage)
P(1,:)=sbest; %randperm(N); %Randomly chooses initial sequence (this will be handled with the SA portion of the code)
s0=P(1,:);

for p=1:N-1
   for q=p+1:N
      x=x+1;
      NS=sbest;
      temp=NS(p);
      NS(p)=NS(q);
      NS(q)=temp;
      %NS(N+1)=NS(1);
      NbhdOfSolns(x,:)=NS;
   end
end



% Use Makespan to calculate the Time
for a=1:NumOfSolns
%% Makespan

P=zeros(2,N); %Creates empty processing sequence table (row1=jobs, row2=completion time after each stage)
```

```
P(1,:)=NbhdOfSolns(a,:); %Randomly chooses initial sequence (this will be handled with the SA portion of the code)
s0=P(1,:);

CompletionTime=zeros(N,M); %Creates empty Makespan table
MachineCT=zeros(1,M); %Table keeps track of completion time for each machine
MachinePN=ones(1,M); %Table keeps track of number of products that have been on each machine
Schedule=zeros(N,M); %Overall schedule of all machines/all stages

%Stage 1

i=1; %current stage

for j = 1:M %For the first product on each machine
   CompletionTime(1,j)=ProcessTime(P(1,j),i); %Completion time is the same as processing time
   Schedule(1,j)=P(1,j); %Keeps track of products on machines
   P(2,j)=CompletionTime(1,j); %Updates current completion time for each product during Stage 1
   MachineCT(j)=CompletionTime(1,j); %Updates current completion time for each machine
end

for j=M+1:N %For the remaining products during stage 1
   [minMCT,machine]=min(MachineCT); %Finds which machine gets the next product
   MachinePN(machine)=MachinePN(machine)+1; %Updates the number of products on each machine
   CompletionTime(MachinePN(machine),machine)=minMCT+ProcessTime(P(1,j),i); %Updates the completion time of the chosen machine
   Schedule(MachinePN(machine),machine)=P(1,j); %Keeps track of products on machines
   P(2,j)=CompletionTime(MachinePN(machine),machine); %Updates current completion time for each product for Stage 1
   MachineCT(machine)=CompletionTime(MachinePN(machine),machine); %Updates current completion time for each machine
end
```

```
CompletionTime( ~any(CompletionTime,2), : ) = [];  %deletes empty rows
Schedule( ~any(Schedule,2), : ) = [];  %deletes empty rows



%Reorders products in ascending order based on completion time
[Y,sortedindex]=sort(P(2,:));
P=P(:,sortedindex);

%Stages 2 through I

for i=2:I %For all stages beyond Stage 1
   CompletionTime=zeros(N,M); %Erases makespan table
   MachineCT=zeros(1,M); %Erases machine completion time table
   MachinePN=ones(1,M); %Erases machine product number table
   Schedule=zeros(N,M); %Ereases overall schedule

   for j = 1:M %For the first product on each machine
   CompletionTime(1,j)=ProcessTime(P(1,j),i)+P(2,j); %The product's completion time from the
previous stage plus processing time
   Schedule(1,j)=P(1,j); %Keeps track of products on machines
   P(2,j)=CompletionTime(1,j); %Updates current completion time for each product for Stage i
   MachineCT(j)=CompletionTime(1,j); %Updates current completion time for each machine
   end

   for j=M+1:N %For the remaining products
   [minMCT,machine]=min(MachineCT); %Finds which machine gets the next product
   MachinePN(machine)=MachinePN(machine)+1; %Updates the number of products on each
machine
   maxTM=max(minMCT,P(2,j)); %Completion time is the maximum of current completion time
of the selected machine or the product's current completion time from the previous stage
   CompletionTime(MachinePN(machine),machine)=maxTM+ProcessTime(P(1,j),i); %Updates
completion time of chosen machine
   Schedule(MachinePN(machine),machine)=P(1,j); %Keeps track of products on machines
```

```
    P(2,j)=CompletionTime(MachinePN(machine),machine); %Updates current completion time
for each product for Stage i
    MachineCT(machine)=CompletionTime(MachinePN(machine),machine); %Updates current
completion time for each machine

    end

    CompletionTime( ~any(CompletionTime,2), : ) = [];  %deletes empty rows
    Schedule( ~any(Schedule,2), : ) = [];  %deletes empty rows



    %Reorders products in ascending order based on completion time
    [Y,sortedindex]=sort(P(2,:));
    P=P(:,sortedindex);

end

%The makespan is the maximum completion time at the end of the last stage
MakeSpan=max(MachineCT);
Time(a)=MakeSpan;




end



MinTime=min(Time);

if MinTime >= zbest
    Continue=0;
else
    sbest=NbhdOfSolns(a,:);
```

```
    zbest=MinTime;
    Continue=1;
end



end

disp('Steepest Descent');
disp(NbhdOfSolns);
disp(['The best schedule is: ' num2str(sbest)]);
disp(['The time for this schedule: ' num2str(zbest) ' time units']);
disp(' ');
```