UNIVERSITY OF VAASA

FACULTY OF TECHNOLOGY

DEPARTMENT OF PRODUCTION

Yashar Ahmadov

JOB SEQUENCING FOR MANUFACTURING PROCESSES WITH SEQUENCE-DEPENDENT CHANGEOVER TIMES

Master's thesis in

Industrial Management

VAASA 2015

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

CL: Clustering

BF: Brute Force

CE: Complete Enumeration

NN: Nearest Neighbor

NP-hard: Non-deterministic Polynomial-time hard

PP: Prima Power (Company)

MES: Manufacturing Execution System

NC: Numerical Control

ERP: Enterprise Resource Planning

SMF: Sheet Metal Forming

UNIVERSITY OF VAASA
Faculty of Technology
Author: Yashar Ahmadov
Topic of the Master's Thesis: Job Scheduling for Processes with Sequence-Dependent Changeover Times
Instructor: Petri Helo
Degree: Master of Science in Economics and Business Administration
Major Subject: Industrial Management
Year of Entering the University: 2013
Year of Completing the Master's Thesis: 2015 **Pages:** 90

ABSTRACT:

Increasing competitiveness in the markets force companies to increase the efficiency in their operations. One way of increasing the efficiency is decreasing the wastes, including the setup times. Setup processes differ in nature depending on the industry and type of manufacturing. For example, companies operating in the Sheet Metal Forming industry face sequence-dependent tool changeovers; i.e. sequencing of jobs affect the total setup time significantly. This increases the complexity of the scheduling problems; thus, effective approaches are required to solve them. Besides the efficiency, another importance of this work is that we include different components of tool changeovers in our calculations.

This thesis work was inspired by a real scheduling problem in the case company. Four different algorithms were considered to solve the sequencing problems with sequence-dependent setup times. We received real data from the customer of the case company and also generated random data set to achieve statistically significant conclusions. Comparative analyses showed that Clustering approach performs better than others. Different sensitivity analyses were conducted, again showing the effectiveness of the Clustering algorithm. So, this algorithm was proposed to the case company to be implemented in their software.

KEYWORDS: sequence-dependent setup, optimization, heuristics, job scheduling, Sheet Metal Forming

1. INTRODUCTION

1.1. Background information

Increasing competitiveness in the business industries forces the companies to decrease their costs by all means. For manufacturing companies, one major component of costs is setup time which they want to eliminate. Reducing the setup times leads to many improvements; increased productivity is one of them (Spence et al. 1987).

In this paper, job scheduling algorithms with sequence-dependent tool changeover times will be discussed. We start with the definitions of the terms "tool changeover" and "setup", because they are related concepts. Setup operation is generally defined as changing manufacturing status from producing one job to another (Zandin 2001: 95). There can be different kinds of setups such as material, tool, and operator setups. "Tool changeover" is a self-explanatory term which is one component of setup, i.e. the operation of changing tools in order to start the manufacturing of a specific product. In this paper, we use these terms interchangeably; both referring to the tool changeover times. "Sequence-dependent changeover" means that the time spent for changeover at previous step affects the time for the current stage. Sometimes it is not only the previous step affecting the setup time, but the whole sequence of jobs preceding the current step is determining the changeover time.

Job scheduling algorithms aim to find sequence of jobs among alternative routes that satisfies certain optimization criteria. Total tardiness, total completion time, total setup costs, makespan and number of late jobs are some of the criteria used for optimization (Hejazi and Saghafian 2005). In this paper, we discuss ways of minimization of total tool changeover time for each workday.

Inspired by the lean philosophy, companies are now differentiating value-adding and non-value-adding activities in their operations. Value-adding activities are the ones that add to the value of the product and customers pay for that. According to the Business Dictionary (2015), setups are non-value adding, because they generate zero return on investment and can be eliminated without damaging the processes. Nowadays, companies are trying to reduce non-value adding activities in order increase their efficiency, productivity and other performance measures. Setup costs constitute important percentage of the lead time. For example Kenechi et al. (1998: 85), have calculated that only 10 percent of the lead time is devoted to pure production in their case company; the rest are non-value adding activities, including setups. This makes firms focus more on this aspect and avoid the high times spent for tool changeovers. Since the competitiveness is increasing in the market, firms try to be more flexible, high setup times are the first thing to be eliminated in this case. Setup time reduction possibilities have been widely researched, especially in the context of lean systems and some ways have been offered in order to reduce the setup times. SMED (Single-Minute Exchange of Dies) is one example of such techniques. This paper is inspired by a real scheduling problem in the Prima Power Company and we focus on finding the best sequence among dozens of

other routing alternatives with the objective of the minimization of tool changeover times.

Several industries face sequence-dependent changeover times, the companies operating in Sheet Metal Forming (SMF) industry being a good example. The main challenge in solving this type of problems is the computational burden. The payoff between acceptable solution and solving time is an important issue to be decided (Nonaka, et al. 2012). Finding the optimal solution may take such a long time that utilizing such methods may not be feasible. To overcome this difficulty, different heuristic rules by variety of authors have been proposed. Those algorithms provide acceptable solutions within a reasonable solving time. The number of jobs that we will be dealing with is on average 10-15 jobs per day, but it is also assumed that the problem size can be as large as 30 jobs per day. Customers are willing to wait 1-2 minutes for the solution; in case if the problem size is big, this can take 3-4 minutes at most. These are the main restrictions that we will keep in mind while searching solutions for the problem.

Some aspects of the problem discussed in this work have not been researched previously. We will be incorporating so-called “accepted tool change”, different tool and station sizes into our models. These issues change the approach to the solution methods. To reduce the computational difficulties, we apply Clustering, Nearest Neighbor and Traveling Salesman methods which give near-optimal results within reasonable solving time.

The outline of this paper is as follows: we will first introduce the case company and the context of the problem. In the second section a brief review of the literature will be given; here, similar works done by various authors will also be discussed. The third section discusses the methodology used in this paper. The following parts of the thesis cover the results and findings of the work. In the last section, we will give conclusion and wrap up the thesis work.

1.1. Case company introduction

The case company Prima Power operates in the Sheet Metal Forming (SMF) industry; produces machines and systems for sheet metal working. SMF industries are initial stage of manufacturing for wide spectrum of industries including automotive, aerospace, energy, domestic appliances, electric cabinets, elevator, escalators etc. Generally they consist of the units as punching, shearing, laser cutting, sheet metal handling systems, etc. SMF industries, compared to the other section of discrete part manufacturing industries, are highly automated and flexible in production diversities.

Along with the machines, Prima Power offers different software to its customers for efficient management of the production. Tulus is one such software family that is used as Manufacturing Execution System (MES) and is fully synchronized with the ERP system of the factory. MES's are used to supervise the process control systems and they also decide the route of the

produced goods (Valckenaers & Van Brussel, 2005). It allows customers to add the orders easily to the task list and manage their manufacturing process. Currently, Tulus does not do the sequencing of jobs in an optimal way; instead First-Come-First-Served (FCFS) principle is applied.

1.2. Problem introduction

Case company wants to optimize the sequencing of jobs within days which will minimize the total time spent on tool changeovers. That is the purpose of this thesis work; optimizing the machine task list by finding the optimal (or near-optimal) arrangement of jobs for each machine.

In this paper, we discuss the application of the proposed algorithms in the case company, but it can be applied for a solution of similar problems in any related manufacturing environment. Before going into details, we introduce the five components involved in the problem.

1. **Machines:** different types of punching & shearing, punching & laser cutting machines that are produced by PP.
2. **Turrets:** a round-shaped structure that holds the tools.
3. **Stations:** a "nest" in the turret where the tools are installed.
4. **Tools:** different types of metal structures that give the specified shape to the products.
5. **Parts (orders):** finished sheet metals.

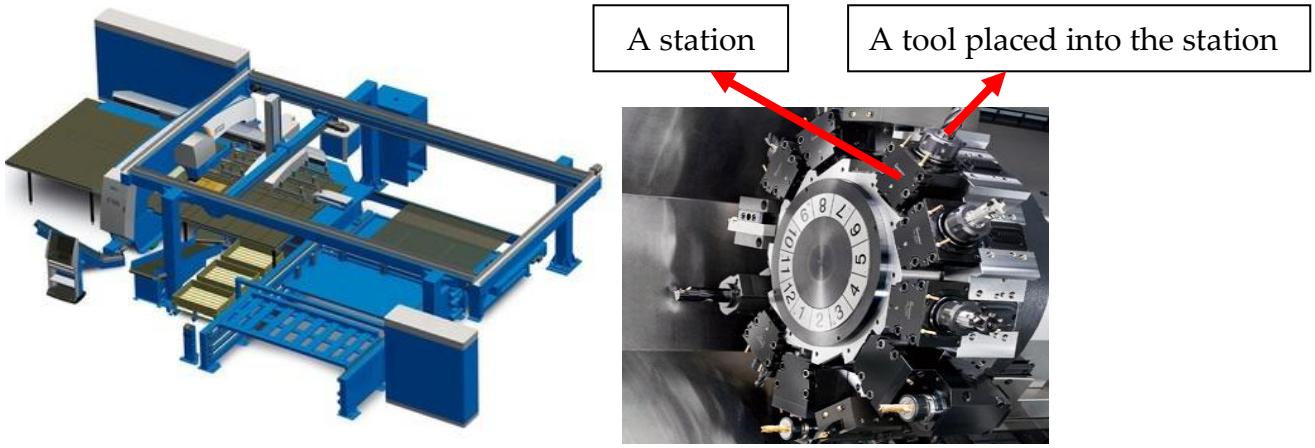


Figure 1. Machine and turret.

Parts to be produced (jobs) are shown in **Figure 2**.

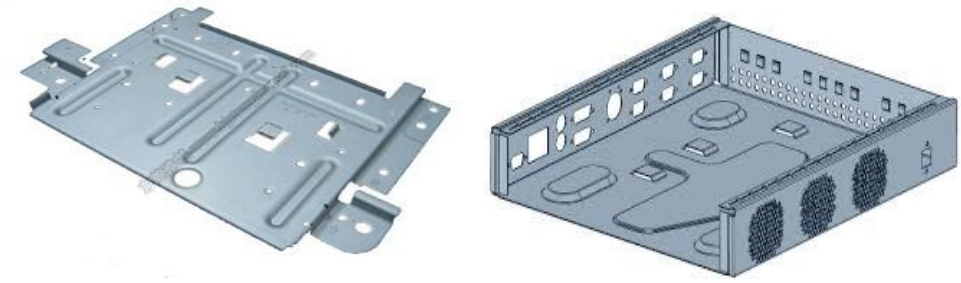


Figure 2. Typical products of SMF industry.

After the general production plan is made, nesting is done for all tasks. The objective of the nesting is to decide which parts will be produced from one sheet metal by reducing the waste of raw materials (Rao et al. 2007: 439). As a result of nesting, required tools, load angles, hits, die clearances are set. In order to manufacture a product, all the necessary tools should be loaded into the turret. **Table 1** exhibits an example of such information for a sample task.

Table 1. Typical Tool, Angle and Die Clearance requirements.

Station	Tool	Load angle	Hits	Die	Size
2	RD7.0	0	4660	0.30	B
7	SQ20.0	0	320	0.20	B i
11	RD20.0	0	180	0.20	D* i
21	SQ5.0	0	3360	0.20	MT6-A *
22	RD2.5	0	860	0.20	MT6-A *
24	RD6.0	0	340	0.20	MT6-A *

The legends of columns are as follows:

Station: Station # in the turret. The stations #1-20 are normal stations, #21-... are multi-tool stations.

Tool: The name and shape of the tool that is assigned to the Station

Load angle: The initial angle of the tool

Hits: The number of punching hits. We do not use this column in our calculations.

Die: The value of the die clearance

Size: The size of the station (“i” stands for indexable)

2. LITERATURE REVIEW

In this part of the paper, we give a review of the literature related to the sequence-dependent setup time reduction. The problem that we are focusing on is a part of scheduling problems and has been investigated since the middle of the 20th century. The abridged form of the problem is FS-SD, where FS refers to Flow Shop and SD to Sequence Dependency. A setup can be needed after a job or batch; the convention FS-SD-job and FS-SD-batch are used to express these two situations. Other variations are also present such as JS (Job Shop), Hybrid Flow Shop (HFS), etc.

According to Burtseva et al. (2010), the total time that a product spends in machine consists of three parts: setup, production and removal. In the past, the companies used to ignore the setup and removal times thinking that they are negligible. But Pinedo (2008) showed that ignoring setup times can decrease the efficiency of the machines for more than 20%. Also he proved that including setups in the scheduling makes the problem NP-hard which is more complex to solve than traditional approach.

Often the problem has been modeled as Traveling Salesman Problem (TSP); various algorithms, MILP linear programming and dynamic programming have been employed to solve the problem. Lockett and Muhlemann's paper (1971) is the most related article to our case. They discuss a scheduling problem with sequence-dependent changeover times. The authors divide the total changeover time into two: first-order serial and higher-order serial. First-order

serial setup is caused due to the previous job and higher-order setups are caused by the jobs before the previous task. Problems with size of up to 35 jobs are solved using various heuristics. Namely, Random Ordering, Traveling salesman without backtracking, Traveling-salesman approach, closest unvisited city algorithms and their performance were tested. The results show that Traveling Salesman with no backtracking dominates the other methods (Lockett and Muhlemann 1971). In their case, it is assumed that jobs require tools in specific stations, i.e. the order of tools in turret are predefined. For example, assume that we have six tools in total and turret has four stations. Further assume that a job requires tools (2, 4, 6, 8) in order to start the manufacturing process. It is not allowed to change the order of tools in turret; the order should be exactly as (2, 4, 6, 8). But in our problem, a tool can be in any stations as long as it fits into the station.

Bowers et al. (1995) offer cluster analysis in order to minimize the sequence-dependent changeover times. Their approach is to group similar jobs, find optimal sequence within groups and then find optimal sequence among groups (**Figure 3**).

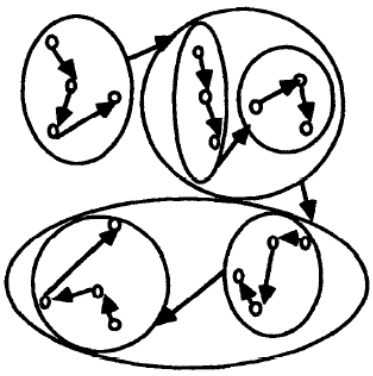


Figure 3. Clustering approach (Bowers 1995: 34).

Thus, near-optimal solutions are achieved; their calculations show that the gap between optimality and heuristic result is less than 5%. This paper prompted me how to decrease the computational effort. The number of possible sequences increases exponentially and an implicit enumeration becomes infeasible. Below is given a graph of number of jobs versus number of possible sequences. The solving time of such problems also increase exponentially; for 15 jobs, number of possible combinations is expressed in terms of 10^{11}, as depicted in **Figure 4**.

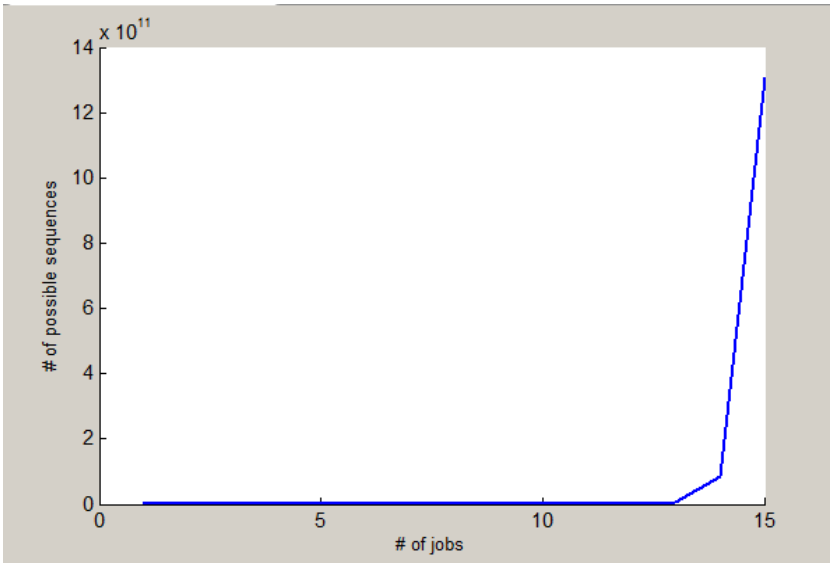


Figure 4. Number of jobs vs possible sequences.

One of the prominent works in this field belongs to Nonaka, et al. (2012). The authors discuss scheduling with alternative routings in CNC workshops. Although sequence-dependency has not been directly taken into account, the paper gives novel ideas about sequencing in similar work environments. Process planning and production scheduling are integrated to increase the efficiency. They combine mathematical optimization and tabu search for finding the optimal route and assigning the operations to machines,

respectively. The problem is similar to our case in the sense that there are x! different routing alternatives for x jobs and this increases dramatically as number of jobs increase. They tackle this difficulty by using the column generation method. The column generation method is an algorithm used for solving huge linear programming models and it has been used successfully in many cases.

Hark Hwang and Ji Ung Sun (1998) discuss sequencing problem with re-entrant work flows and sequence-dependent setup times. Re-entrant work flow means that jobs are performed in a specific machine more than one time during the sequence of the manufacturing process. Scheduling problem consists of n jobs that are to be produced in two machines and the objective is the minimization of the makespan. The authors employ modified dynamic programming to solve the problem and find the optimal sequence.

An interesting approach by White and Wilson (1977) is worth mentioning. They employ regression model to find out and classify the significant factors that affect setups for each machine. 93 setups have been collected by setup personnel for this purpose. Then the regression model is used to predict the setup times for sequence-dependent operations. Regression model unveils hidden characteristics of the setup operations which is important for sequencing. Next stage is sequencing of tasks using predicted changeover time values for which different optimization tools and heurists were employed. Authors model the problem as a Traveling Salesman Problem (TSP) with no requirement to return to origin, which has NxN cost (or distance, time) matrix.

The advantage of the solution heuristics offered in the paper is that they are easy and do not take much time to solve.

Gupta (1982) offers branch and bound method to solve scheduling problems with sequence-dependent setup for n jobs and m machines. His objective function was minimizing the total setup cost. But again, he assumes that the setup time of switching from job A to B is the same as switching from B to A which is not a valid assumption in our case. One other drawback of his algorithm is that it is limited to small problems, i.e. it is not efficient in solving large problems.

Mirabi (2011) proposes a modified Ant Colony Optimization (ACO) algorithm to solve the Sequence-Dependent setup problems with the objective of minimizing the total makespan. His findings show that new algorithm performs better than regular ACO algorithm.

3. METHODOLOGY

This section of the thesis work discusses the methodologies considered for finding solutions to the case problem. The algorithms used in this thesis were coded and run in the MATLAB software.

3.1. Complete Enumeration Method (or Brute Force algorithm)

Complete enumeration method considers all possible permutations (sequences) and finds the optimal solution. This method guarantees that optimal solution will be found, however it might be costly from solution time point of view. To illustrate the method, we give the following example:

There are three tasks and we are aiming to find the sequence of jobs that will result in the least total completion time. Then we have 3!=6 alternative routes:

3	2	1
3	1	2
2	3	1
2	1	3
1	2	3
1	3	2

We consider each of the routes; calculate the total tool changeover time. Then, the sequence that yields the minimum time is selected as optimal sequence. This method is sometimes called as “brute force” in the literature.

3.2. Nearest neighbor algorithm

Nearest neighbor (NN) algorithm is one of the earliest methods for solving several problems in Operations Research, including the TSP-model problems. We start with one data point, then find the nearest neighbor and continue this process until all points have been covered. This algorithm does not provide an optimal solution, but the main advantage is that it is much faster compared to the optimization methods. If the solution is within an acceptable range, then NN algorithm can be preferred due to its fastness. The pseudo code for this algorithm can be shown as follows:

1. Pick one data point, X
2. Find the nearest unvisited data point, Y
3. Set current point to Y
4. Mark Y as visited
5. Stop if all data points have been visited, otherwise, go to step 2.

At this point, the important question is that how the "nearest" distance is measured. There are different measures proposed in literature about this issue. For our problem, there is one criterion i.e. the distance is one dimensional. We are interested in only the total setup time for a set of tasks and the "nearness" will be measured according to the setup time between operations. To explain the algorithm, consider the following simple example. The initial point is A, the set {B, C, D, E, F} represents the points to be visited and the numbers show the distances between the possible routes.

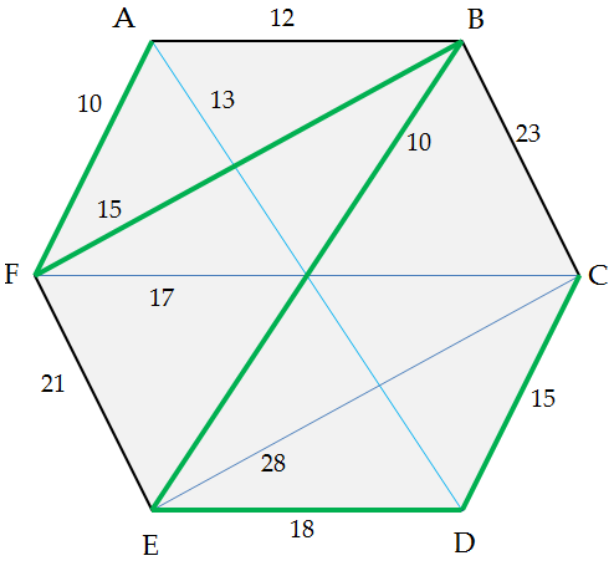


Figure 5. Nearest Neighbor approach.

We start at A and have three options: {B, D, F} with respective distances {12, 13, 10}. We choose F, because it has the minimum distance, 10; and this point is market as visited. From F, we there are three available points {B, C, E} with distances {15, 17, 21}, respectively. So, we choose B because it is the closest unvisited point. This procedure continues until all points are visited; in our case the resulting route is A-F-B-E-D-C as shown in **Figure 5**.

3.3. Nearest neighbor clustering

Since the implementation of Complete Enumeration method for sample sizes larger than six is practically infeasible, a heuristic was developed that combines Clustering and CE. Many authors have used this approach in the literature, for

example Von Luxburg et al. (1981) have explained the applications of such algorithms for discrete optimization problems. Clustering is grouping data points that share certain characteristics and there are different types and ways of clustering. Some clustering approaches allow a data point to be included in several clusters, others do not. Some clustering approaches use probability, i.e. the data points belong to some group with certain probability. (Ian Witten et al. 2005).

k-nearest neighbors says that a data point belongs to the same group with its k closest points. In simple words it means '*do what your neighbors do*' (Adriaans et al. 1996: 56-57). Again the 'closeness' can be defined in many ways, in our case this is the time spent for tool changeovers. The algorithm works as follows:

1. Find 5 closest points to the current point
2. Optimize the sequence for those 5 jobs
3. Repeat this procedure until all jobs are sequenced

3.4. Traveling Salesman Problem (open tour)

Traveling Salesman Problem is an important part of the Operations Research science. Given a set of cities (or points), the distances between them, TSP aims to find the shortest route. These types of problems are NP (non-deterministic polynomial-time) hard and become challenging to solve when there are dozens of cities in the data set (Papadimitrou 1971).

The TSP problem is formulated as shown in **Figure 6**.

$x_{ij} = \begin{cases} 1 \text{ if the path goes to city j from city i} \\ 0, \text{ otherwise} \end{cases}$

$$\min \sum_{i=0}^{n} \sum_{j \neq i, j=0}^{n} c_{ij} x_{ij}$$

$$0 \leq x_{ij} \leq 1 \qquad i, j = 0, \cdots, n \qquad (1)$$

$$u_i \in \mathbf{Z} \qquad i = 0, \cdots, n$$

$$\sum_{i=0, i \neq j}^{n} x_{ij} = 1 \qquad j = 0, \cdots, n \qquad (2)$$

$$\sum_{j=0, j \neq i}^{n} x_{ij} = 1 \qquad i = 0, \cdots, n \qquad (3)$$

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad 1 \leq i \neq j \leq n \qquad (4)$$

Figure 6. TSP formulation.

First constraint defines the binary variables x_{ij}. Second constraint ensures that each city is visited from only one city; third constraint assures that the path goes to only one city. Last constraint is for preventing the sub-tours, i.e. the optimal solution consists of one complete tour. In our case, the "cities" will be the jobs and "distance" will be the setup time required to switch from one job to another.

3.5. Data collection

Data set 1 consists of a task list for two days with 6 and 19 jobs. It was generated as an example by the company's workers.

Data set 2 is a real data obtained from a customer of Prima Power which is operating in the Vaasa region of Finland. It includes 13 days' task list for the machine LPE6 with LSR6 robot which is connected to automatic sheet storage made by the company Fastems. LPE is a combi machine with laser and punching features. The summary of the data set is given in **Table 2**.

Table 2. The structure of the Data Set 2.

Date	No. of jobs
10.09.2014	2
11.09.2014	4
12.09.2014	4
16.09.2014	4
19.09.2014	3
22.09.2014	3
23.09.2014	4
25.09.2014	1
26.09.2014	4
27.09.2014	5
29.09.2014	15
30.09.2014	3
01.10.2014	2

The days with one and two jobs were combined with the preceding days' data because the solutions for such days can be found easily without using optimization algorithms.

Data set 3 consists of 200 day's orders and was randomly generated using MATLAB.

Data set 4 was received from one of the customers of Prima Power. It contains all the orders for the first quarter of the year 2015. More detailed description is given in the **Table 3**. Again, the days with less than 5-6 jobs were combined with the next day in order to challenge the algorithms.

Table 3. The structure of the Data Set 3.

Day	No. of jobs		
	Jan	**Feb**	**Mar**
1	4	3	12
2	3	10	2
3	7	3	6
4	6	2	2
5	19	15	11
6	14	9	13
7	3	5	13
8	4	8	7
9	8	3	3
10	4	2	13
11	7	2	5
12	6	7	10
13	12	-	2
14	5	-	4
15	11	-	-

4. FORMULATION AND DISCUSSIONS

In this section of the thesis, we explain the results and findings of this research. So far, we have explained the problem, discussed methodology and have given a review of the works done in this field. Now that the problem is defined, the next steps are formulation and solution.

4.1. Problem formulation

At the beginning, it is important to clarify the inputs and outputs of the problem. The inputs of the problem are:

a. Jobs, their tool, angle clearance requirements
b. Turret, its capacity, station sizes, initial conditions
c. All tool numbers, their sizes
d. Average time spent for tool, clearance, angle changes and adapter plugging

And the expected outputs are:

a. Optimal sequencing of jobs
b. What tools to change at which stage, where to install them in the turret (this output is optional)

Objective and constraints of the problem are given below:

Objective: **MINIMIZE** Total tool changeover time = (1) tool change time + (2) adapter plugging time + (3) die clearance change time + (4) angle change time

Subject to the constraints/requirements:

- Each job has its tool requirements. These tools should be in turret while processing the order.
- Turret's station sizes define what tools can be fitted into them.
- Tools are also of different sizes.
- Changing tools in turret takes some time (1).
- A tool can be fitted to a turret station with larger size. This needs an adapter, which takes some time (2) to attach to the tool.
- Clearance values (material thicknesses) should be taken into account. The tools should have respective die clearance, if any tool is with wrong clearance, it should be corrected. This change takes some time (3).
- Each task has its specific initial angle requirements, i.e., tools should be at pre-defined angles when the manufacturing process starts. Any angle change takes time (4).

In this paper, we solve the scheduling problems for only one machine at a time.

4.2. An Example

This section discusses an example of how the setup procedure is conducted in the SMF industry. Also, the data input/output for the MATLAB models are explained. Let's assume that when we start the workday there are tools number 1 and 4 already available in the turret. The initial configuration is visualized in **Figure 7**. Different sizes of circles represent the stations with different sizes.

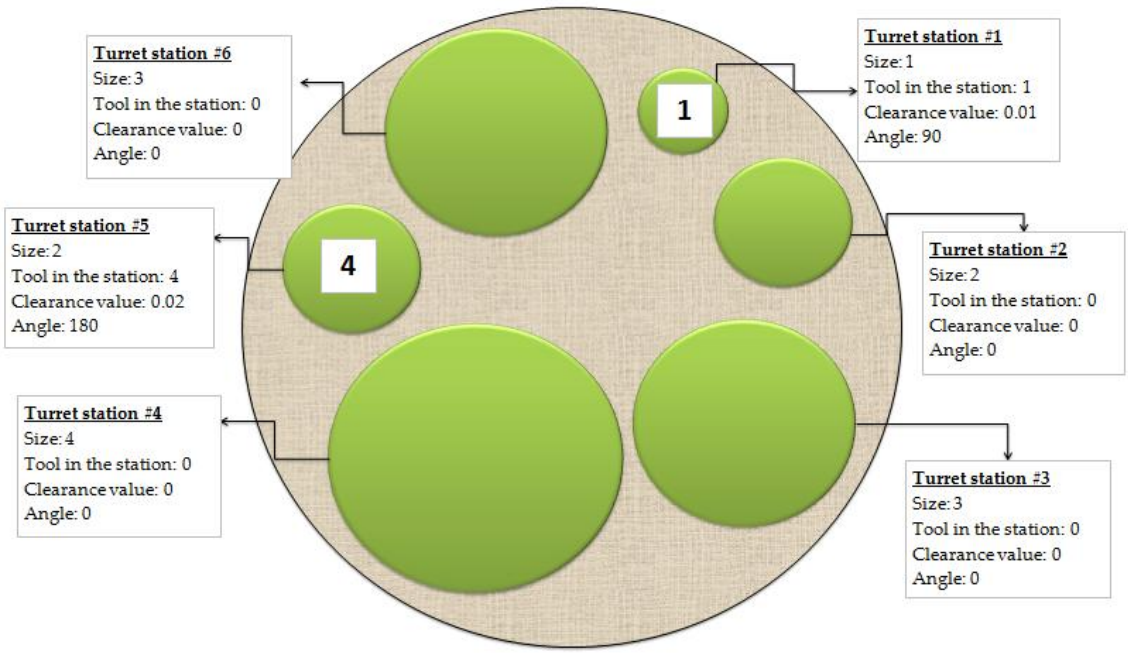


Figure 7. A sample turret.

During the workday, the company has to complete seven jobs and each job requires tools with corresponding angle and clearance values as given in **Table 4**.

Table 4. Tool, Angle, Die Clearance requirements.

Job #	Tool requirements	Tools' angles	Tools' clearances
1	[1 2 3 5 6 7]	[90 0 0 180 270 0]	[0.01 0.05 0.02 0.08 0.03 0.04]
2	[2 3 4 8]	[0 90 180 270]	[0.02 0.04 0.05 0.03]
3	[1 2 5]	[0 90 0]	[0.02 0.07 0.02]
4	[1 2 3 5 6 7]	[90 0 0 270 180 360]	[0.01 0.02 0.03 0.04 0.03 0.02]
5	[1 2 5 8]	[0 90 0 180]	[0.02 0.08 0.09 0.01]
6	[1 3 6]	[270 90 180]	[0.02 0.08 0.09]

The sizes of the vectors in the 2nd, 3rd and 4th columns should be same because each tool has a load angle and clearance value. The information in the above table is recorded in the parameter "*Job*" which has struct data type and it keeps information about jobs. For example, the first job is entered into the system as *Job(1).ToolRequirements* =

[1 90 0.01
2 0 0.05
3 0 0.02
5 180 0.08
6 270 0.03
7 0 0.04]

In general, the first column of the matrix *Job(X).ToolRequirements* represents the *ToolNumbers*; i.e. tool requirements of the job X. The second and third columns contain *Angles* and *Clearance* values, which represent the angle and clearance requirements of the job X, respectively.

We need to input the stations' sizes and initial condition of stations in turret. Parameter "*capacity*" is the capacity of the turret, i.e. number of the stations in turret, which are 6 in our example. Another input "*turret*" also has structure data type and it includes the condition of turret, i.e. which tools are in turret at a given time. Structure data types are coded using the "struct" command in MATLAB. MATLAB's struct command allows the user to store different types of data under one variable.

turret(X).ToolNumber: Tools currently in turret. Zero (0) means that currently there is no tool at the station.

turret(X).Turret_Size: The size of the stations in the turret.

turret(X).Clearance: Corresponding clearance values of the tools in turret

turret(X).Angle: Corresponding angle values of the tools in the turret

In the example case, initial conditions are as follows:

```
turret.ToolNumber=[1 0 0 0 4 0];
turret.Turret_size=[1 2 3 4 2 3];
turret.Clearance=[0.01 0 0 0 0.02 0];
turret.Angle=[90 0 0 0 180 0];
```

Next input is about the tools in general and their corresponding sizes. "*tools*" is a vector for tool numbers and "*tool_size*" contains the corresponding sizes of the tools. In this case, we assume that there are 8 tools to be utilized in the whole production process. The second row contains the sizes of the tools, size 1 being the smallest and 4 the largest:

```
tools=[1 2 3 4 5 6 7 8];
tool_size=[1 3 2 2 4 1 1 2];
```

Total time spent for setups is divided into four parts:

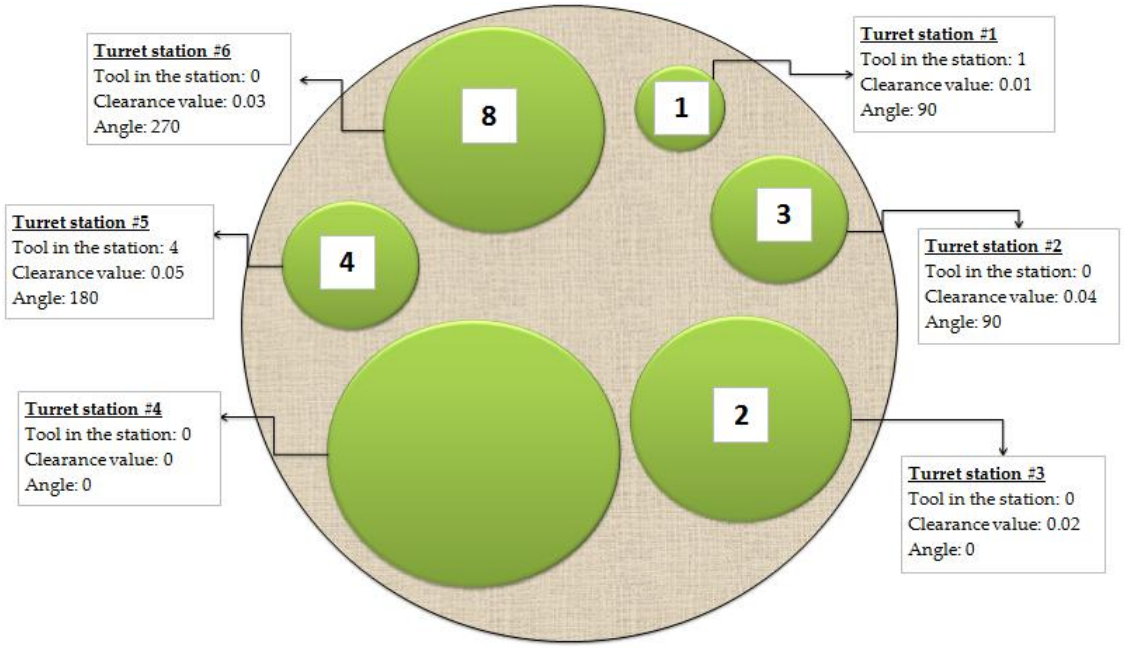
1. Time spent for tool change
2. Time spent for changing the die clearance
3. Time spent for changing the angle of the tool
4. Time spent for plugging in the adapter

Here we assumed that tool change time is 5 min/tool, die clearance change time is 2 min/tool, angle change takes 1 min/tool and adapter plugging time is 3 min/tool. Inputs *tool_change_time*, *clearance_change_time*, *angle_change_time* and *adap_plug_time* are defined for this purpose.

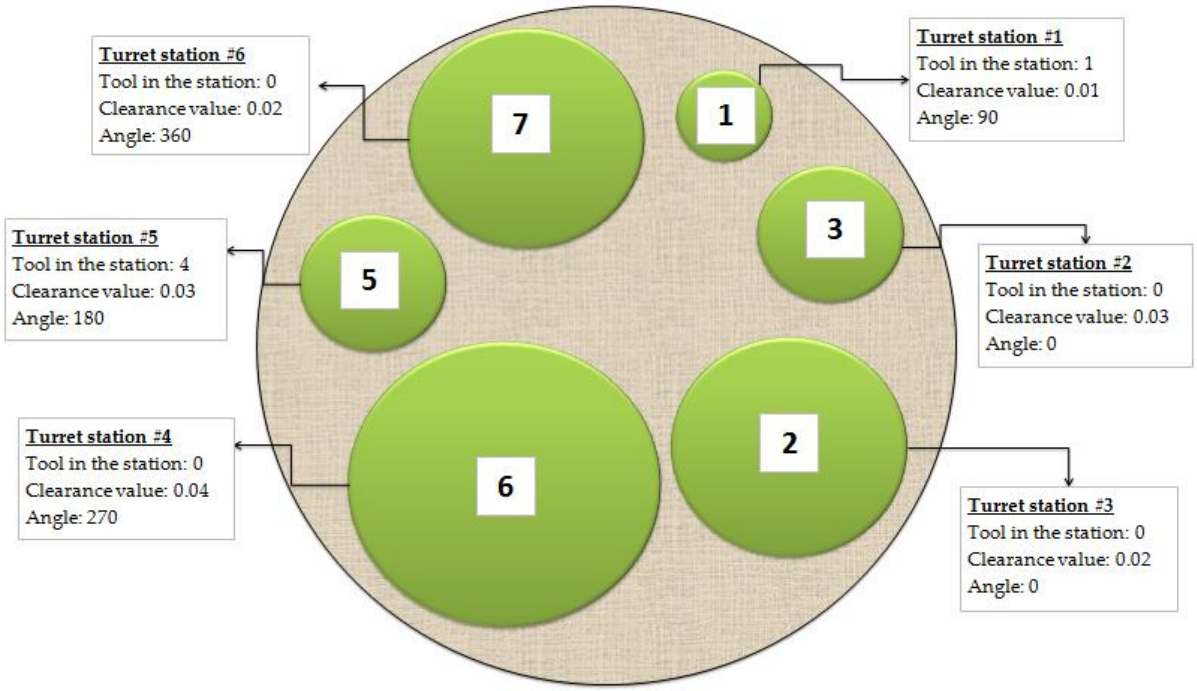
```
tool_change_time=5;
clearance_change_time=2;
angle_change_time=1;
adap_plug_time=3;
```

Now let's suppose that we are going to process the jobs in the following order 2-4-1-6-3-5. This is the optimal sequence which will be explained in the upcoming sections. The following drawings (**Figure 8**) exhibit the evolution of the turret during the process:

Step 1: Processing job #2



Step 2: Processing job #4



.

.

.

.

Step 6: Processing job #5

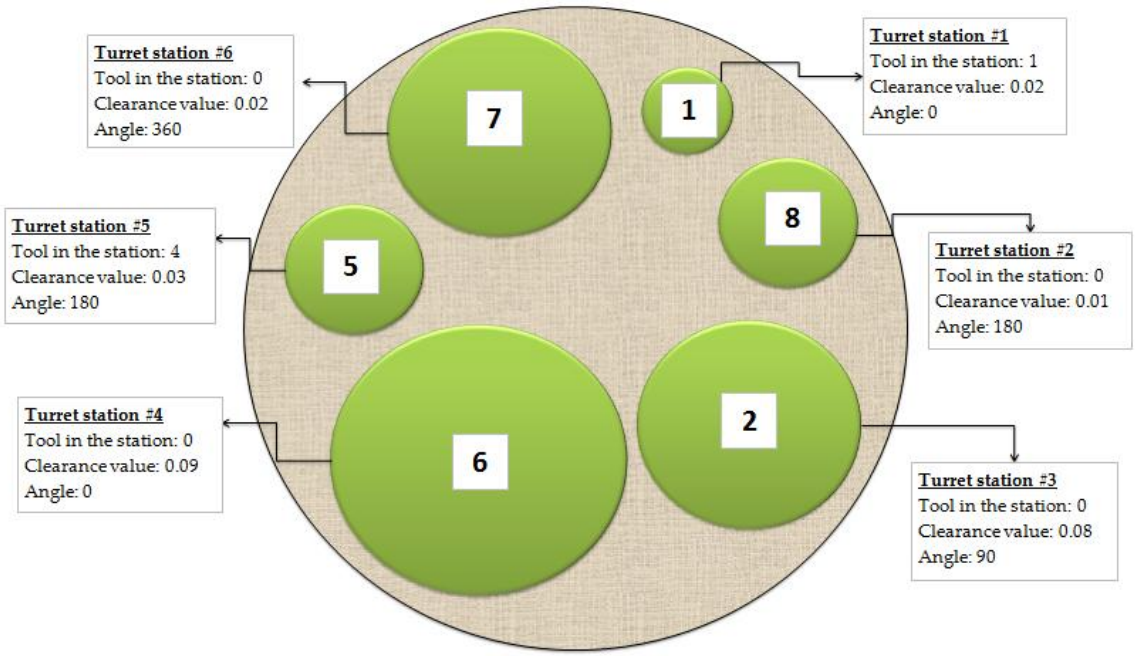


Figure 8. Changes in the turret.

Now let's have a closer look at the changes in turret when we switch from stage 1 to stage 2.

a) The tools 1, 3 and 2 stay in the turret, on the other hand, tools 5, 6 and 7 are added. This means that there are three tool changes; considering that installing each tool takes 5 minutes, then 3x5=15 minutes is spent on these installations.

b) As noted above, the tools 1, 3 and 2 stay in the turret when switching from stage 1 to 2. But in stage 2, the tool 3's clearance value is different from stage 1 (0.04 and 0.03, respectively), that is why, die clearance change is needed. 1x2=2 minutes will be spent for this process.

c) The same is valid for the angles. Again, there is a change in the load angle of tool 3; it should be rotated from 90° to 0°. 1x1 = 1 minute is needed for this change.

d) The last component of tool setups happens when a tool with smaller size is installed to a larger station. In our example case, the tool number 7 is installed in station number 6. But the tool is smaller than the size of the station and we need to use an adapter while placing the tool. This takes 1x3 =3 minutes.

Adding up all these four numbers, we get 21 minutes (15+2+1+3). This is the cost of processing the job #4 after job #2. Our aim is to find the sequence that results in minimum time spent for setups. The code (which we will discuss in the upcoming sections) considers all possible combinations of jobs, calculates setup times for each of them and gives the optimal sequencing.

```
JobSequence =

     2     4     1     6     3     5


SetupTime =

    73
```

The output gives the minimum number of tool changes and the optimal order of the jobs. It means that job #2 should be done first, then job #4, then #1, #6 and so on. If we follow this order, we will spend 73 minutes for setups, which is the minimum possible value.

The MATLAB code can also provide an optional output about what tools to change at each step and in which stations to install them. **Figure 9** depicts the Gantt chart for the optimal solution of our example case.

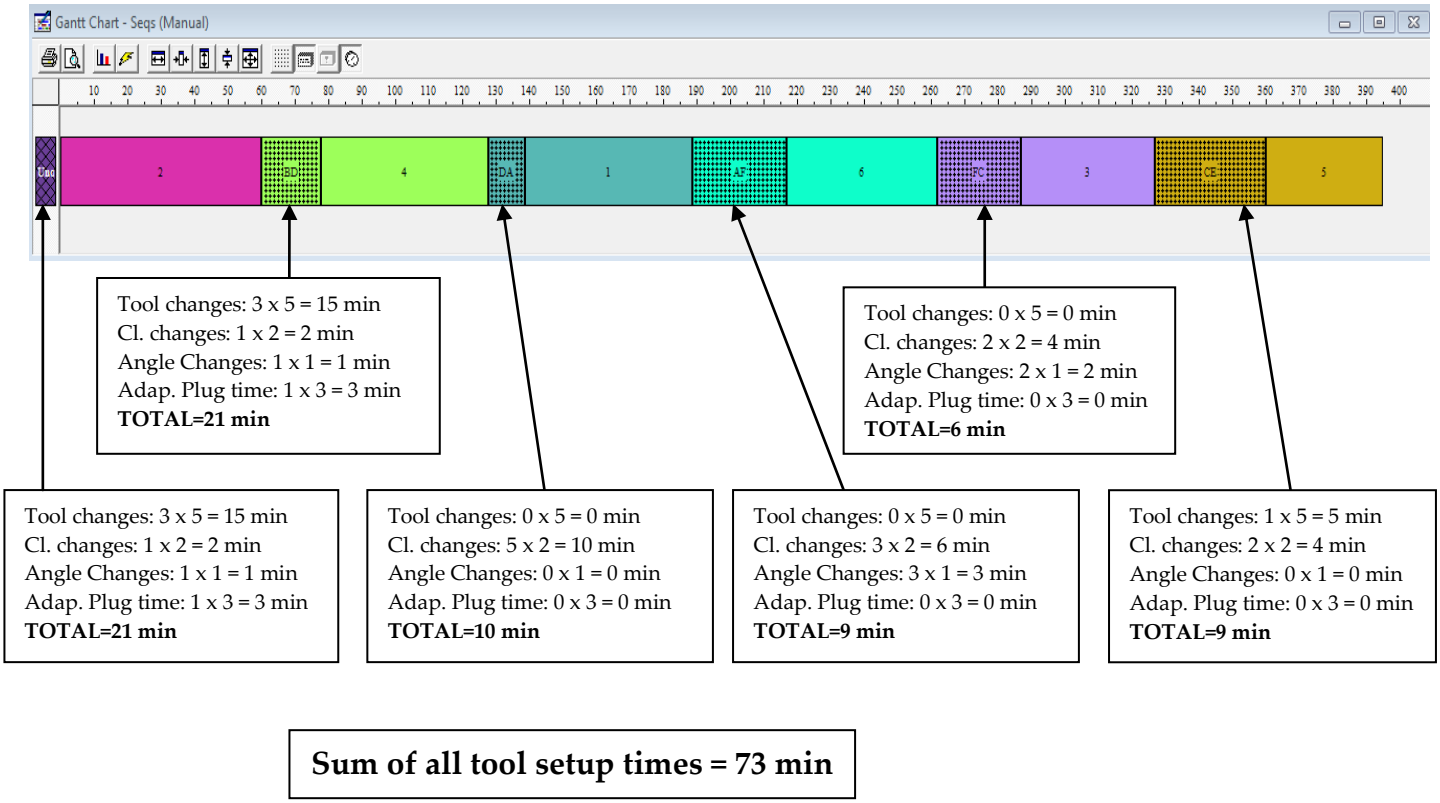


Figure 9. The Gantt Chart of the production process.

The distribution of the different components of the tool changeovers is depicted in **Figure 10**.

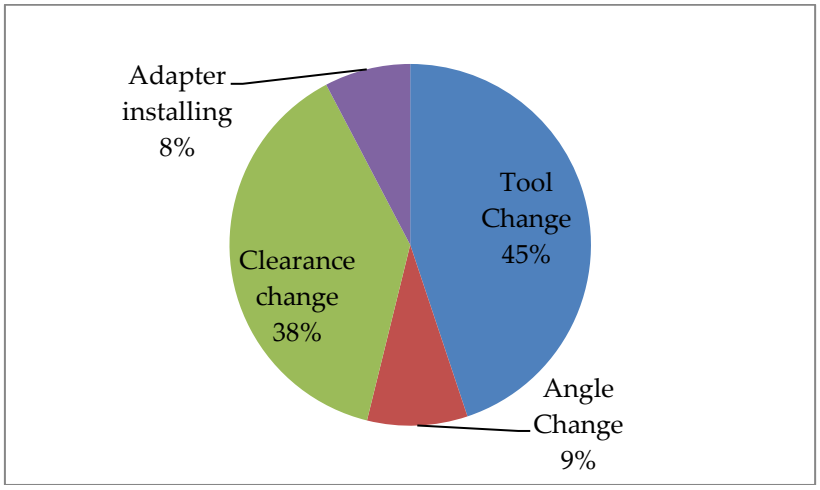


Figure 10. Pie Chart of the distribution of the setup times.

As can be seen from the chart, pure tool changes take only half of the total time. The rest of the time is devoted to angle, clearance and die changes. This thesis work is also important from this aspect; we take into consideration the other factors that affect the total tool changeover time.

4.3. Some discussions before solving

Now that the problem is modeled, we can start searching for the solution. The first function programmed in MATLAB gives the total changeover time for a sequence given as input. Since finding the optimal point in such combinatorial problems is practically infeasible from solution time point of view, we need to use some heuristics. Many heuristics and search methods such as Genetic Algorithms are nothing but clever guessers. Before elaborating on advanced

methods, we can visualize the jobs versus the tools required to have some initial ideas (**Figure 11**).

Jobs\Tools	2	3	4	5	6	7	8	9	12	13	14	15	16	17	19	22	24	25	27	28
1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1
2	0	0	0	0	0	1	1	1	0	1	0	0	1	0	0	0	1	0	0	1
3	1	0	0	0	0	1	1	1	0	1	0	0	1	0	0	0	1	0	0	1
4	1	1	1	0	0	1	1	1	0	1	0	0	1	1	0	0	1	0	1	1
5	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	1	1
6	1	0	0	0	0	1	1	1	0	1	1	0	1	0	0	0	1	0	1	1
7	1	0	0	0	0	1	1	1	0	1	1	0	1	0	0	0	1	0	1	1
8	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	1	0	0	1
9	0	0	0	0	1	1	0	1	0	1	1	1	1	0	0	0	1	0	1	1
10	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	1	0	1	1
11	1	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0	1	0	1	1
12	1	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0	1	0	1	1
13	1	0	1	0	0	1	1	1	0	1	0	0	1	1	0	0	1	0	0	1
14	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	0	1	0	1	1
15	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	1	1
16	1	0	0	0	0	1	1	1	0	1	1	0	1	0	0	0	1	0	1	1
17	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
18	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	1
19	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	0	1	0	1

Figure 11. Jobs vs Tool Requirements.

Without using those advanced tools, the operators can themselves too do the guessing. As it can be seen from the **Figure 11**, some jobs use the same tools, others not. For example Jobs 6 & 7, 11 & 12 require totally the same tools, so, these jobs can be processed sequentially. If we start with the job number 15, for the next jobs we will have to change the tools only a few times because it contains majority of the tools needed for processing all the jobs. This data package consists of 19 jobs, which is far above the average but for data sets with less than 6-7 jobs, it is fairly easy to guess.

4.4. Algorithms

After doing a thorough literature review, we came up with four different approaches to the solution of our case problem. First, the pseudo codes were prepared and then were programmed in MATLAB. This section introduces the pseudo codes of the algorithms.

The following signs and functions are used in the pseudo-codes:

*length(a): The length of the array a.

*horzcat(a,b): horizontal concatenation of vectors a and b.

*a\b: The set difference of a and b.

* ∅: Empty set.

*min(x): The smallest element of the vector.

*last(a): The last element of the vector a.

4.4.1. Complete Enumeration (Brute Force)

0: Take inputs

1: Generate all possible routing alternatives

2: **for** q=all possible sequences

- **for** m=all sequential pairs [i-1, i] in the sequence q
  - Calculate # $of\ tool\ changes$

- Calculate # $of\ clearance\ changes$
- Calculate # $of\ angle\ changes$
- Calculate # $of\ adapter\ pluggings$

- **end for**
- Add up all the number of tool changes by category:

$$Total\ \#\ of\ tool\ changes(q) = \sum_{m}(\#\ of\ tool\ changes\ (m))$$

$$Total\ \#\ of\ clearance\ changes(q) = \sum_{m}(\#\ of\ clearance\ changes\ (m))$$

$$Total\ \#\ of\ angle\ changes(q) = \sum_{m}(\#\ of\ angle\ changes\ (m))$$

$$Total\ \#\ of\ adapter\ plugging(q) = \sum_{m}(\#\ of\ adapter\ pluggings\ (m))$$

3: **end for**

4: Calculate total time for each sequence q:

$$\begin{aligned} Total\_time(q) = {} & Total\ \#\ of\ tool\ changes(q) * tool\_change\_time \\ & + Total\ \#\ of\ clearance\ changes(q) * clearance\_change\_time \\ & + Total\ \#\ of\ angle\ changes(q) * angle\_change\_time \\ & + Total\ \#\ of\ adapter\ pluggings(q) * adap\_plug\_time \end{aligned}$$

5: Find the sequence q that corresponds to min(Total_time)

In simple words, the code takes inputs and generates all possible sequences of the jobs. For example, with 3 jobs, there will be 3!=6 different alternate routes. For the sequence 1-2-3, the total number of setups is calculated in two steps: for

switching from 1 to 2 and from 2 to 3. The 2nd part of the code does this for all the sequences. Once we get the total setup numbers for all sequences, the part 4 multiplies those numbers of setups with the equivalent times that each process takes. The last line chooses the route with the minimum time.

4.4.2. Nearest Neighbor

0: Take inputs

1: Set current_point = 1, visited ={1}, unvisited={2,3,…n}

2: **for** q=1:length(unvisited)

Calculate the following values for the [current_point, unvisited(q)]

- $\#\ of\ tool\ changes$
- $\#\ of\ clearance\ changes$
- $\#\ of\ angle\ changes$
- $\#\ of\ adapter\ pluggings$

3: **end for**

4: Calculate total time for each sequence q:

$$Total\_time(q) = \#\ of\ tool\ changes(q) * tool\_change\_time + \#\ of\ clearance\ changes(q) * clearance\_change\_time + \#\ of\ angle\ changes(q) * angle\_change\_time + \#\ of\ adapter\ plugging(q) * adap\_plug\_time$$

5: Find the next point x that has min(Total_time)

6: Set current_point = x, visited=horzcat(visited,x) , unvisited=unvisited\X

7: If unvisited= Ø, stop; the solution is the vector "visited". Otherwise, go to step 2.

In this code, lines 0 and 1 serve for taking inputs and initialization. Parts 2, 3 and 4 calculate the number of tool changes and the time for switching from the current job to all of the unprocessed jobs. For example, if we are now manufacturing the first product, that part of the code will calculate the switching cost from 1 to 2, from 1 to 3, from 1 to 4 and so on. The fifth line chooses the minimum of those costs. Let's assume that switching from job #1 to #3 takes the lowest time, then, the vector *visited* will be {1,3}, and *unvisited* will be {2,4,5,…}. This loop continues until *unvisited* becomes an empty set. The final order of set elements of *visited* is the solution of the Nearest Neighbor algorithm.

4.4.3. Clustering

0: Take inputs

1: Calculate the distance_matrix

- The distance_matrix is nxn matrix that contains the tool changeover times for switching from one job to another. The diagonals are zero, since processing the same job sequentially would not need any tool changes.

2: Set current_point = 1, visited ={1}, unvisited={2,3,…n}

3: Find the five closest points X according to the distance_matrix

4: Apply Complete Enumeration algorithm and find the optimal sequence for those five jobs, label the output as X'

5: Set visited =horzcat(visited, X'), current_point = last(visited)

6: Set unvisited=unvisited\X

7: Update the distance matrix; place 1000 to the columns that represent the already visited points.

8: If unvisited= Ø, stop. The solution is the vector visited. Otherwise, go to step 3.

The first part of this algorithm calculates the distance matrix; the second part initializes the variables. The next parts choose the 5 jobs that will need minimum tool change from the current point. Section 4 applies the CE algorithm to find the optimal sequence of jobs for those 5 jobs. The columns of the *distance_matrix* corresponding to the scheduled jobs are changed to 1000. This ensures that already visited points will not be selected again. The algorithm continues this process until all points are visited. Once completed, the vector *visited* contains the near-optimal route of the manufacturing.

4.4.4. TSP Model

In order to apply the TSP model, several simplifications were introduced:

1. Among different type of changeover components, only number of tool changes is considered.
2. Only the effect of the preceding job on the current job is considered. The higher order changeovers are ignored.

We build a distance matrix and continue the calculations based on that. In fact, this is a modified version of the CL approach; the difference is that the sequencing of jobs are done using TSP linear model. More precisely, the algorithm works as following:

0: Take inputs

1: Calculate the distance_matrix

2: Set current_point = 1, visited ={1}, unvisited={2,3,…n}

3: Find the five closest points X according to the distance_matrix

4: Apply TSP approach and find the optimal sequence for those five jobs, let's say the output is the set X'

5: Set visited =horzcat(visited, X'), current_point = last element of "visited"

6: Set unvisited=unvisited\X

7: Update the distance matrix; place 1000 to the columns that represent the already visited points. (This ensures that the visited points will not be visited again.)

8: If unvisited= Ø, stop. The solution is the vector visited. Otherwise, go to step 3.

After explaining the four different approaches employed to solve the case problem, now we proceed to the findings of the work. The algorithms were applied to the data sets that were explained in the methodology part.

5. RESULTS

In this section of the paper, we discuss the findings of the thesis work. The results are grouped according to the data sets. First we start with data set 1, compare the algorithms when applied on this specific data. Then, the real data set is analyzed in order to find the algorithm that performs better than others. The same procedure is applied for the data sets 3 and 4 to replicate the findings. Sensitivity analysis is discussed at the end of the section.

5.1. Data Set 1

This data set consisting of randomly generated two days' orders was used for warm-up purposes and test for possible bugs. The algorithms gave expected results; also some minor bugs in the codes were found and corrected. The CL algorithm was not used for the first day, since it needs more than 6 jobs to get reasonable results. When applying the three different algorithms in MATLAB, we get the following results (**Figures 12** and **13**).

Alg.	**Sequence of jobs**
CE	6-4-5-2-3-1
NN	4-3-6-5-1-2
TSP	4-3-2-1-6-5
CL	N/A

Figure 12. The order of jobs for Day 1.

Alg.	Sequence of Jobs
CE	N/A
NN	1-17-18-2-3-13-4-8-15-5-7-12-16-6-9-11-10-14-19
TSP	17-18-19-5-6-7-16-4-13-3-2-1-8-10-14-9-12-11-15
CL	13-4-15-11-12-14-9-16-7-6-1-5-2-3-10-18-17-8-19

Figure 13. The order of the jobs for Day 2.

The total setup times according to the different algorithms are summarized in **Table 5**.

Table 5. Total setup times.

Day	# of jobs	CE	NN	TSP	CL
1	6	42	47	48	N/A
2	19	N/A	78	103	89

As can be seen from the **Table 5**, TSP gives the worst results in both cases. We noted at the beginning that several assumptions (simplifications) were introduced in order to apply the TSP method. But the company does not want to use this approach just for the sake of simplicity. Complete enumeration gives the optimal results but it is not applicable for large problems due to high solving time. Now, we are left with two approaches: Nearest Neighbor and Clustering.

Comparing NN and Clustering

In order to get basic ideas about the performance of these two algorithms, we used the Day 2's data. Every time we add one job and solve the problem using both NN and CL approaches. Graphs number 8 and 9 show the results. **Figure 14** depicts the solution values (i.e. total setup time) versus number of jobs. Up to 14 jobs, Clustering gives better results, but after that NN suppresses Clustering. But the data is not large enough to draw exact statistical conclusions about the performances of these algorithms.

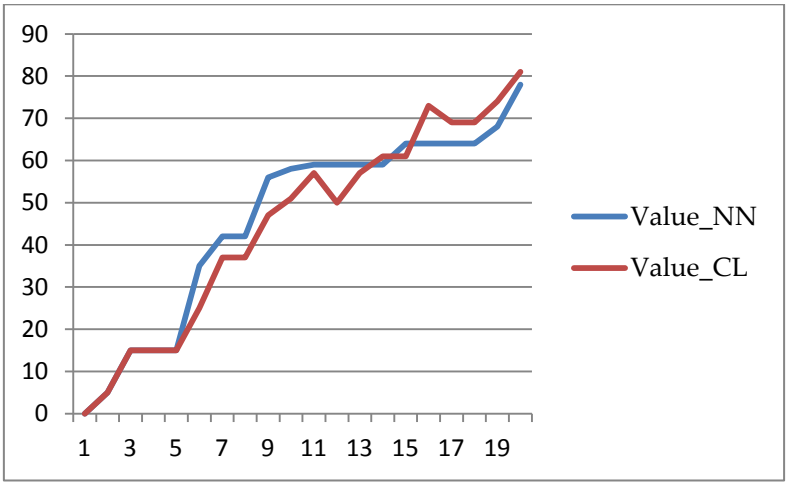


Figure 14. Solutions for NN and CL.

Figure 15 depicts solution time (in seconds) versus number of jobs. Both algorithms' solution time increases linearly, but Clustering increases faster than NN. Still, none of them exceed 20 seconds, which is an acceptable value.

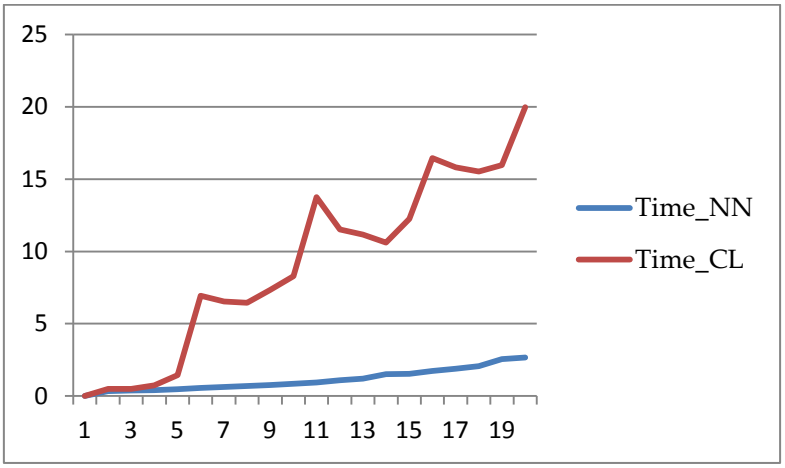
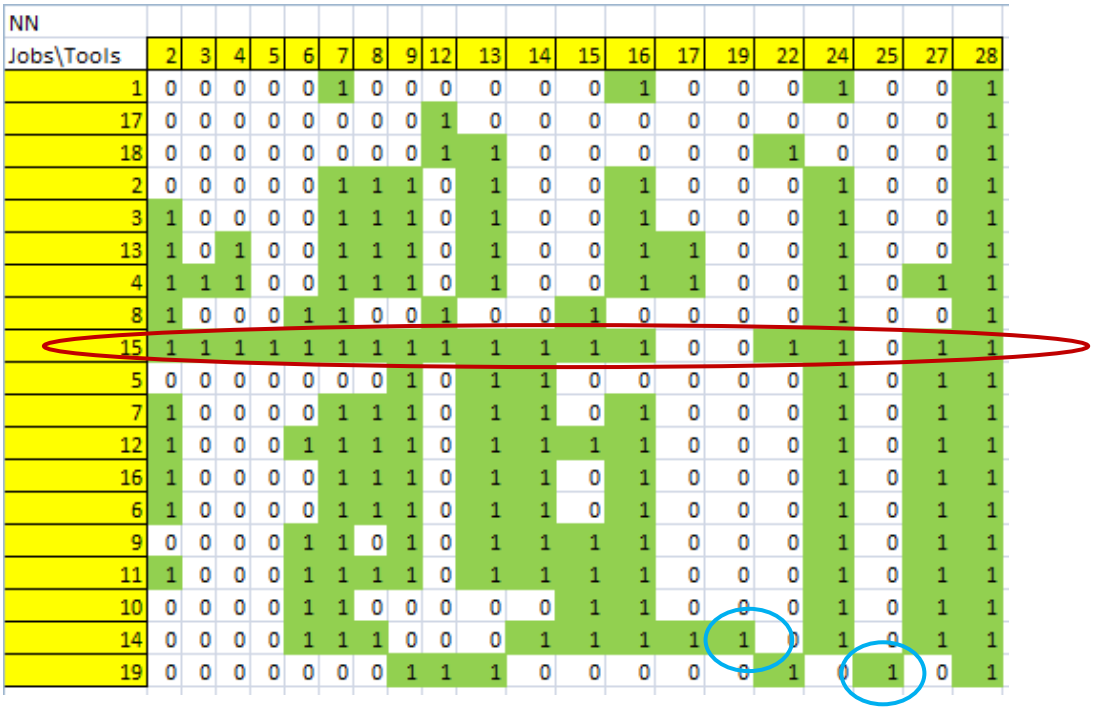


Figure 15. Solving times for NN and CL (sec).

It is difficult to decide between them by analyzing just one sample. We will come back to this issue in the next sections.

By visualizing the solutions in matrices for the three approaches, we can get more insight about their way of working. Below is given the solution of the Nearest Neighbor algorithm (**Table 6**).

Table 6. Tools matrix of the NN algorithm.



NN																				
Jobs\Tools	2	3	4	5	6	7	8	9	12	13	14	15	16	17	19	22	24	25	27	28
1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1
17	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
18	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	1
2	0	0	0	0	0	1	1	1	0	1	0	0	1	0	0	0	1	0	0	1
3	1	0	0	0	0	1	1	1	0	1	0	0	1	0	0	0	1	0	0	1
13	1	0	1	0	0	1	1	1	0	1	0	0	1	1	0	0	1	0	0	1
4	1	1	1	0	0	1	1	1	0	1	0	0	1	1	0	0	1	0	1	1
8	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	1	0	0	1
15	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	1	1
5	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	1	1
7	1	0	0	0	0	1	1	1	0	1	1	0	1	0	0	0	1	0	1	1
12	1	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0	1	0	1	1
16	1	0	0	0	0	1	1	1	0	1	1	0	1	0	0	0	1	0	1	1
6	1	0	0	0	0	1	1	1	0	1	1	0	1	0	0	0	1	0	1	1
9	0	0	0	0	1	1	0	1	0	1	1	1	1	0	0	0	1	0	1	1
11	1	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0	1	0	1	1
10	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	1	0	1	1
14	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	0	1	0	1	1
19	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	0	1	0	1

The solution is logical; we start with job #1, and then add some tools every time until we reach the job #15 (marked with red). For the rest of the jobs, we need to change just a few tools. More precisely, for jobs #14 and #15 we might need to add 3 tools (marked with blue), but if they are already in the turret, no change is needed.

The solution of the clustering algorithm is shown in **Table 7**.

Table 7. Tools matrix of the CL algorithm.

Jobs\Tools	2	3	4	5	6	7	8	9	12	13	14	15	16	17	19	22	24	25	27	28
13	1	0	1	0	0	1	1	1	0	1	0	0	1	1	0	0	1	0	0	1
4	1	1	1	0	0	1	1	1	0	1	0	0	1	1	0	0	1	0	1	1
15	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	1	1
11	1	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0	1	0	1	1
12	1	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0	1	0	1	1
14	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	0	1	0	1	1
9	0	0	0	0	1	1	0	1	0	1	1	1	1	0	0	0	1	0	1	1
16	1	0	0	0	0	1	1	1	0	1	1	0	1	0	0	0	1	0	1	1
7	1	0	0	0	0	1	1	1	0	1	1	0	1	0	0	0	1	0	1	1
6	1	0	0	0	0	1	1	1	0	1	1	0	1	0	0	0	1	0	1	1
1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1
5	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	1	1
2	0	0	0	0	0	1	1	1	0	1	0	0	1	0	0	0	1	0	0	1
3	1	0	0	0	0	1	1	1	0	1	0	0	1	0	0	0	1	0	0	1
10	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	1	0	1	1
18	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	1
17	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
8	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	1	0	0	1
19	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	0	1	0	1

The solution of the TSP algorithm is depicted in **Table 8**.

Table 8. Tools matrix of the TSP algorithm.

Jobs\Tools	2	3	4	5	6	7	8	9	12	13	14	15	16	17	19	22	24	25	27	28
17	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
18	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	1
19	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	0	1	0	1
5	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	1	1
6	1	0	0	0	0	1	1	1	0	1	1	0	1	0	0	0	1	0	1	1
7	1	0	0	0	0	1	1	1	0	1	1	0	1	0	0	0	1	0	1	1
16	1	0	0	0	0	1	1	1	0	1	1	0	1	0	0	0	1	0	1	1
4	1	1	1	0	0	1	1	1	0	1	0	0	1	1	0	0	1	0	1	1
13	1	0	1	0	0	1	1	1	0	1	0	0	1	1	0	0	1	0	0	1
3	1	0	0	0	0	1	1	1	0	1	0	0	1	0	0	0	1	0	0	1
2	0	0	0	0	0	1	1	1	0	1	0	0	1	0	0	0	1	0	0	1
1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1
8	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	1	0	0	1
10	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	1	0	1	1
14	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	0	1	0	1	1
9	0	0	0	0	1	1	0	1	0	1	1	1	1	0	0	0	1	0	1	1
12	1	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0	1	0	1	1
11	1	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0	1	0	1	1
15	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	1	1

It can be seen from the matrix that TSP algorithm starts with the job that needs the smallest number of tools and ends with the one that requires the most number of tools.

By analyzing the tool frequency chart (**Figure 16**), it becomes obvious that some tools are used more frequently than others. For example, Tool number 28 is required for processing all orders; on the other hand, tool number 19 is utilized few times.

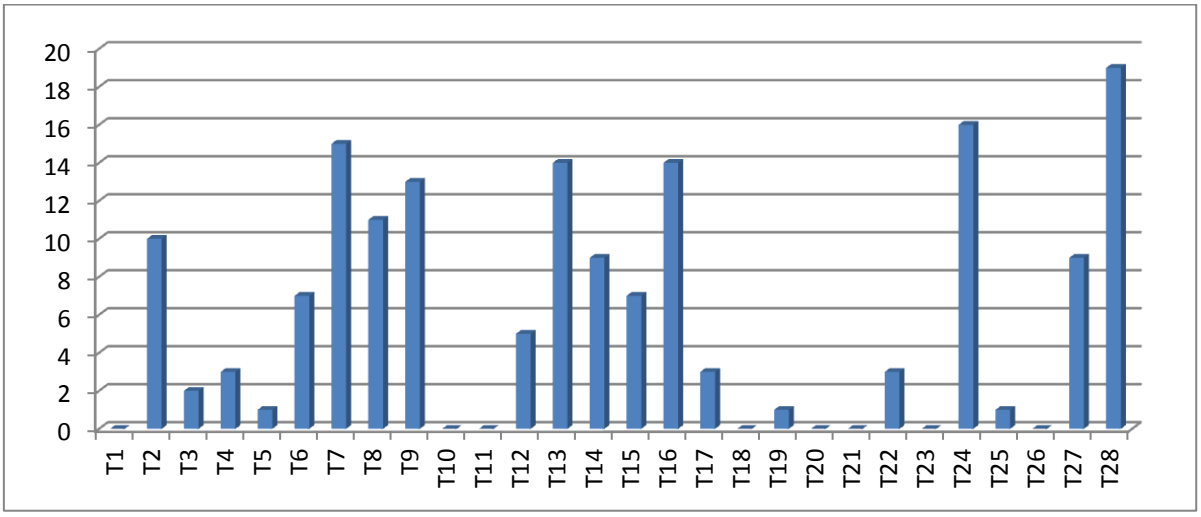


Figure 16. Utilization frequency of the tools.

We also tested one simple algorithm based on the frequency of the tools:

1. Sort tools columns according to SUM() descending

2. Sort jobs rows according to SUM() descending

The resulting order of the jobs is the proposed solution. If we apply these steps to the above-mentioned data, we get the matrix shown in **Table 9**.

Table 9. Tools matrix for the heuristic approach.

	28	24	7	13	16	9	8	27	2	14	6	15	12	4	17	22	3	5	19	25	1	10	11	18	20	21	23	26
15	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0
11	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	1	1	1	0	1	0	1	1	0	1	1	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
6	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	1	1	1	1	1	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	1	1	1	1	1	1	1	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
16	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	1	1	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	1	1	1	0	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	1	0	0	1	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0
1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
17	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The columns are the tools and rows are the jobs. So, the resulting sequence is 15, 4, 11, 12…17.

The initial purpose of implementing this procedure was to find out whether there are some clusters in the data that can be differentiated. But by analyzing the matrix, it was concluded that there are no real clusters in our data set. The resulting matrix can be interpreted as follows: fill in all the turret stations with the tools at the beginning of the day. Then only a few changes may be needed during the day. However, this does not guarantee the optimality.

5.2. Data Set 2 (From Customer Company)

In this part of the thesis, the results of the real data set analyses are discussed. We first give some descriptive information about the data set; below is given the frequency diagram of the needed tools (**Figure 17**).

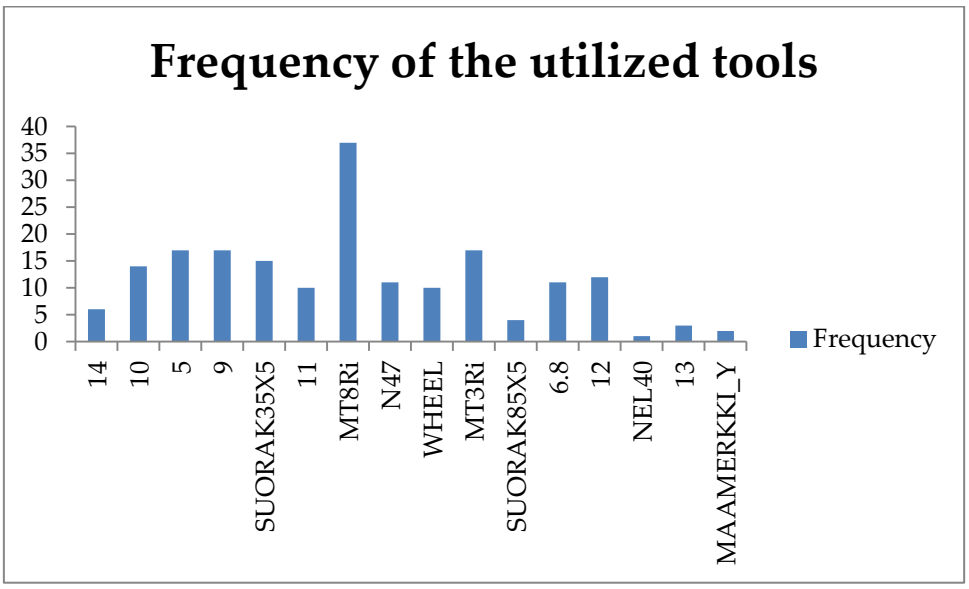


Figure 17. The frequency of the utilized tools.

In general, 16 different tools were utilized for the jobs altogether. As can be seen from the above diagram, the multi-tool MT8Ri is needed for the majority of the jobs. Some others, like NEL40 are used only a few times. The turret has 20 stations and the initial configuration is as shown in **Figure 18**.

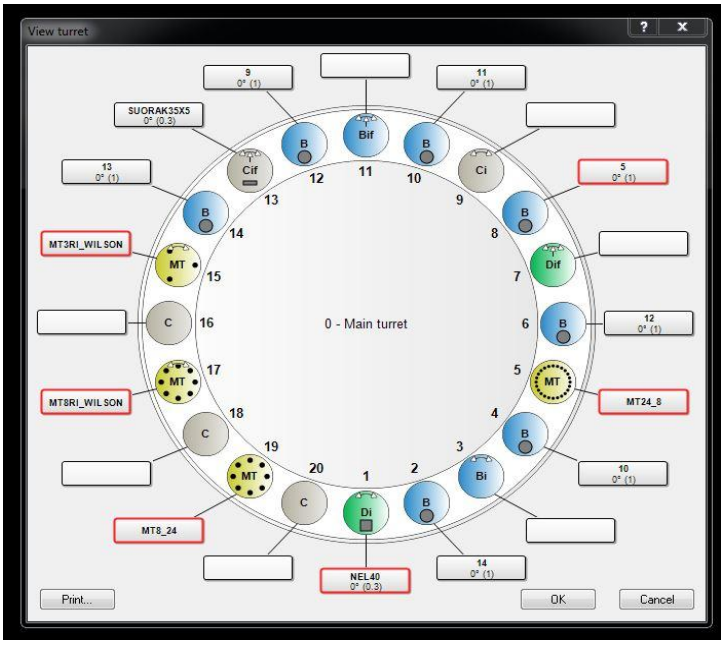


Figure 18. Initial turret configuration.

Here, ABCDE are normal stations and F, G,...,R are different types of multitools. Four out of twenty stations are multi-tool stations. Two of the multi-tool stations (numbers 15 and 17) are indexable, the other two (5 and 19) are non-indexable (fixed). For example, MT8_24 is a fixed multitool in the station 19; it is bolted to turret permanently. Only multitool stations that include letter "i" are drop-in multitools and they can be installed to Di-stations. Multitools which name include R (rotation) means that they are indexable.

The fixed multi-tools are hard to disassemble and load new tools. Drop-in multi tools always have to be in indexable stations because they need to be rotated to select the correct tool inside them. Indexable means that tool can rotate to programmed angle freely. The rotation coordinate comes from NC-coordinate with C-axis command.

The summary of the outcomes of the algorithms are compared in **Table 10**.

Table 10. The solutions and solving times.

Day	No. of jobs	NN sequence	NN time	Cluster Sequence	Cluster time	TSP Sequence	TSP Time
1	2	123	0	231	0	213	0
2	4	1432	16	3241	16	4231	16
3	4	1234	14	4312	14	1432	14
4	4	4123	14	2413	14	4213	14
5	3	231	24	213	24	321	24
6	3	123	7	231	7	321	7
7	4	2341	4	4321	4	3421	4
8	4	2134	6	4312	6	1342	6
9	6	163452	23	165432	23	621534	21
10	15	1 5 13 11 8 6 12 9 2 3 4 7 10 14 15	44	10 8 5 3 2 14 13 12 7 4 15 11 9 1 6	34	8 14 13 10 4 3 2 7 12 5 6 9 11 15 1	34
11	5	45132	17	4 3 5 2 1	17	3 1 5 4 2	17

Day: the number of the day

No. of jobs: number of jobs for the day

NN sequence: The resulting sequence according to the Nearest Neighbor algorithm

NN time: The total setup time according to the results of NN

Cluster Sequence: The resulting sequence according to the Clustering algorithm

Cluster time: The total setup time according to the results of Clustering

TSP Sequence: The best sequence according to TSP

TSP Time: The total setup time according to the results of TSP

Based on these figures, the Clustering algorithm performs better than the others. We used the worst possible sequence for benchmarking. The results show that the optimization algorithms give better results as the number of jobs increase. The days with 2-6 jobs are simple; they can even be sequenced by the operator. As the complexity increases, the optimization algorithms help more. **Table 11** gives the comparison of the Clustering algorithm results with the

benchmarking values. In three cases, clustering algorithm decreases the setup time by 11%, 25% and 23%.

Table 11. The solution according to the CL approach.

Day	No. of jobs	Cluster time	Benchmarking value	% better
1	2	0	0	-
2	4	16	18	11%
3	4	14	14	-
4	4	14	14	-
5	3	24	24	-
6	3	7	7	-
7	4	4	4	-
8	4	6	8	25%
9	6	23	23	-
10	15	34	44	23%
11	5	17	17	-

With 2-3 jobs, the algorithms do not help much. To overcome this difficulty, we offer two ways:

1. The jobs can be combined as long as they meet the due date. For example, on day 1, there are 2 jobs and on the next day, we have 4 jobs. They could be combined (as long as this does not violate the due date constraint) so that we have 6 jobs. Then we can apply optimization methods.
2. The second way is that when there are less than 5 jobs, we do not apply these optimization methods. Instead, the workers can decide themselves easily with the help of the visualization discussed on page 37.

The solution times of the algorithms are all within reasonable range as shown in **Figure 19**. TSP takes significantly more time than the others, but it does not exceed 30 seconds.

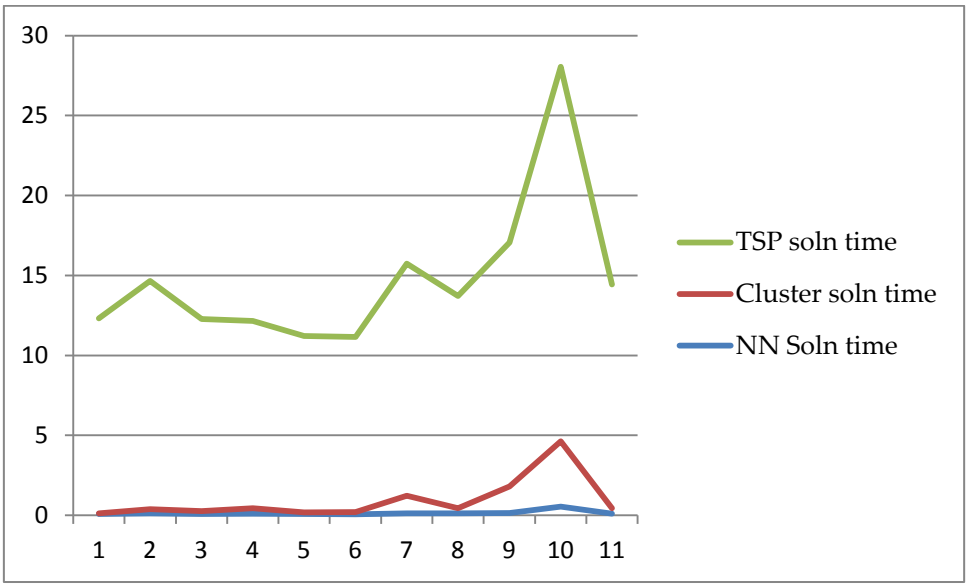


Figure 19. Code running times of the algorithms.

One last point is that in this real data case, we had 16 tools in total and turret had 20 stations. So, the number of tools is less than the number of the stations in the turret. But by definition, the setup time reduction algorithms are more helpful in the cases when there are far more tools than the turret stations.

5.3. Data Set 3 (Randomly Generated Data Set)

For this part of thesis, a random sample with orders for 200 days was generated. The MATLAB code for this process is given in **Figure 20**.

```
for i=1:200                    % For each day i from 1 to 200
    x = randi([1 30]);         % Generate x jobs
    clear Test_job1
    for j=1:x                  % For each job j
        y=randi([1 10]);       % Generate number of required tools
        z1=unique(randi([1 30],y,1), 'stable'); % Generate tool numbers
        angles=[0 45 90 135 180 225 270 315 360];
        pos1 = randi([1 9],length(z1),1);  % Choose one of the possible angles randomly
        z2 = angles(pos1);
        clearances=[0 0.1 0.2 0.3 0.4 0.5];
        pos2= randi([1 6],length(z1),1);    % Choose one of the possible die clearance values randomly
        z3=clearances(pos2);
     % Assign the generated Required tool numbers, Angles and Clearances
        Test_job1(j).ToolRequirements(:,1)=z1;
        Test_job1(j).ToolRequirements(:,2)=z2;
        Test_job1(j).ToolRequirements(:,3)=z3;
    end
```

Figure 20. MATLAB code of the random data generation.

Then the generated samples were solved using the Nearest Neighbor, Clustering and TSP algorithms. An extract from the Excel file is given in **Table 12**.

Table 12. An extract from the generated random data.

No of jobs	NN	CL	TSP	DEF	NN-CL	CL-TSP	NN-TSP
11	192	207	196	209	-15	11	-4
13	196	219	215	214	-23	4	-19
12	206	200	204	214	6	-4	2
24	343	357	354	421	-14	3	-11
23	342	328	312	390	14	16	30
15	294	262	294	311	32	-32	0
4	80	73	74	79	7	-1	6
10	132	132	136	132	0	-4	-4
25	460	377	424	443	83	-47	36
28	406	408	414	428	-2	-6	-8
7	144	147	163	151	-3	-16	-19
28	466	446	465	534	20	-19	1
20	281	282	306	320	-1	-24	-25
24	308	315	383	399	-7	-68	-75
5	102	103	122	125	-1	-19	-20

The first column shows the number of jobs for each day, second column is the total setup time if NN is used. The following two columns show the total setup time if Clustering and TSP approaches are employed for solving the same problem. The column "DEF" is the total setup time if the jobs are performed in the order of 1-2-3…n. This is our benchmarking value; because currently the company mainly uses this default order to process the orders. The last three columns display the differences NN-CL, CL-TSP and NN-TSP, respectively. For example, the first row is Day 1 and we have 11 different orders to be fulfilled. If we employ the sequence generated by NN approach, 192 minutes will be spent for setup. On the other hand, if CL algorithm is used, the setups will take 207 minutes. The difference is 15 min; i.e. by using the NN algorithm, we will save 15 min.

A simple analysis shows that CL performs better; because, for 126 out of 200 days, CL gives better solutions than NN. They perform equally for 11 days and for the remaining 63 days, NN's solutions are better. But this is not enough to draw conclusions about the performances of the two algorithms. The most preferred statistical approach is using the paired t-test to check whether the samples are different or not. The main condition of the test is that the data should be normally distributed. As it can be seen from the below histograms (**Figure 21**), the data does not meet this condition.

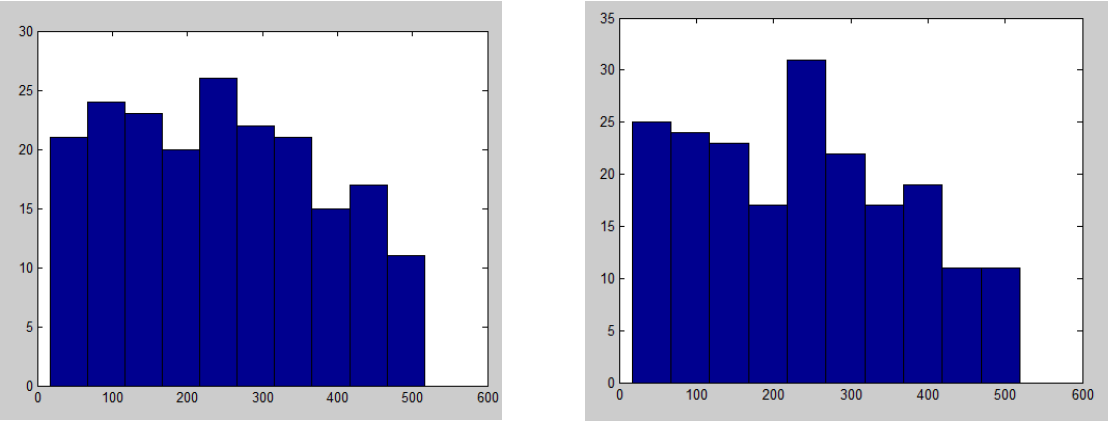


Figure 21. Histograms of the total setup times for NN and CL.

So, we needed to search for alternative tests. In such cases, sign test can be used; it does not require normality assumption. The null hypothesis of the sign test is that the median of the difference (NN-CL) is zero:

H_0: Median of the difference NN-CL=0

H_1: Median of the difference NN-CL≠0

After applying the test to our data set, the resulting p-value is **1.72×10^{-6}**, so we reject the null hypothesis that the median of the difference is zero. This simply means that Clustering and Nearest Neighbor algorithms perform differently. To decide which of these two approaches yields a better result, we test the following hypothesis:

H_0: Median of the difference NN-CL=0

H_1: Median of the difference NN-CL>0

The p-value of this test is 4.22×10^{-18} that is why, the null hypothesis is rejected. The conclusion is that the difference NN-CL is greater than 0, which simply means that applying CL algorithm results in lower setup times.

This can be confirmed by analyzing the descriptive statistics of the "NN-CL" column (**Table 13**).

Table 13. Descriptive statistics of the difference.

NN-CL	
Mean	6.355
Standard Error	1.35624
Median	6
Mode	11
Standard Deviation	19.1802
Sample Variance	367.878
Kurtosis	0.29882
Skewness	0.04218
Range	111
Minimum	-48
Maximum	63
Sum	1271
Count	200
Confidence Level(95.0%)	2.67445

The mean of the difference is 6.355 and median is 6. Thus, since the NN-CL difference is on average more than zero and the confidence interval is on the positive side we conclude that **Clustering** yields better results than Nearest Neighbor. This is because, for example, if NN-CL=5, it means that the total setup time of the route generated by CL is 5 minutes less than NN. That is how we decide about whether a solution is better than another or not.

Following the same procedures for the TSP-CL difference, we get the p-value of 1.08×10^{-24}, meaning that CL performs better than TSP also. The outputs of the sign test are given in **Figure 22**, which indicates that CL dominates the other algorithms.

NN-CL

```
p =

   4.2224e-18


h =

     1


stats =

    zval: 8.5934
    sign: 158
```

TSP-CL

```
p =

   1.0767e-24


h =

     1


stats =

    zval: 10.1920
    sign: 163
```

Figure 22. Sign test statistics for NN-CL and TSP-CL.

While analyzing the first data set, we saw that up to 14 jobs per day, Clustering algorithm gave better results. There were doubts about whether there is such a threshold value that sets the border between two algorithms. It is not statistically correct to decide by analyzing only one data set. Now that 200 random days' data is available, we can test the hypothesis. The test is built as following:

H_0: For the days with more than 14 jobs, median of the difference NN-CL=0

H_1: For the days with more than 14 jobs, median of the difference NN-CL>0

The p-value of the test is **2.74×10^{-8}**, which means that we reject the null hypothesis. Again, we conclude that Clustering performs better than Nearest Neighbor approach. Testing for other possible threshold values did not give consistent results either; CL dominates NN in all cases.

5.4. Data Set 4

After testing the proposed algorithms on randomly generated data set, the results were validated using the real data set 4. Columns *NN* and *CL* represent the total changeover times for NN and CL algorithms, respectively. *DIFF* is the difference of the columns NN and CL. *DEF* is the total changeover time if the jobs are processed using the default sequence, i.e. in the order of Job 1, Job2,…Job n. The results are summarized in **Table 14**.

Table 14. Comparison of the algorithms.

Day	No. of Jobs	NN	CL	DIFF	DEF	CL % better than DEF
1	14	41	30	11	34	11.8
2	25	50	52	-2	58	10.3
3	17	67	63	4	75	16.0
4	12	50	56	-6	59	5.1
5	11	51	52	-1	54	3.7
6	18	40	43	-3	53	18.9
7	16	39	45	-6	49	8.2
8	16	84	81	3	82	1.2
9	17	84	90	-6	107	15.9
10	14	87	89	-2	105	15.2
11	11	76	85	-9	96	11.5
12	11	79	79	0	87	9.2
13	14	63	54	9	64	15.6
14	19	86	76	10	104	26.9
15	13	73	70	3	73	4.1
16	13	29	45	-16	45	0.0
17	23	85	76	9	82	7.3
18	21	102	92	10	98	6.1
					Avg	10.4

The average of the column DIFF is positive, proving our findings in the previous section. On average, CL sequencing saves 10.4 % of the setup time compared to the default sequencing that is currently used.

5.5. Sensitivity analyses

Sensitivity analysis is an important part of any optimization problem. In our case, we discuss how the algorithms behave when i) tool changes, ii) angle changes and iii) clearance changes dominate. The idea is to test what happens when there are a lot of tool, angle or clearance changes. In practice this means that some companies might receive such orders that they use the same set of tools to manufacture the parts. But majority of setup time might be devoted to the angle changes, the rest might be ignorable.

i) When tool changes dominate:

The hypothesis test is built as following:

H_0: Median of the difference NN-CL=0

H_1: Median of the difference NN-CL>0

The p-value is **4.22×10^{-28}**, the null hypothesis is rejected.

H_0: Median of the difference TSP-CL=0

H_1: Median of the difference TSP-CL>0

The p-value is ≈0; again there is no evidence to support the null hypothesis. Clustering works better than NN and TSP.

ii) When angle changes dominate:

To test what would happen if the production process faces a lot of angle changes, we modified the first data set such that all jobs require almost the same tools, but there are lots of angle changes. The comparative results of different algorithms are given in **Figure 23**.

```
Sequence_NN =

    17    18    19    13     4     8    15     1     2     3     5    16     7    12     6    10     9    11    14

total_time_NN =

   125

Sequence_Cluster =

     4     1    15     3     2     7     6    12     9    11    14    13     5    16    10    18    17     8    19

total_time_Cluster =

   136

Sequence_TSP =

    17    18    19     5    16     7     6    11    12     9    14    10     8    13     4     3     2    15     1

total_time_TSP =

   149
```

Figure 23. Extract from the MATLAB output.

The total changeover time is 125 for NN, 136 for CL and 149 for TSP. Again, Nearest Neighbor algorithm provides better solution than the others. This was also proven by using the large sample data:

H_0: Median of the difference NN-CL=0

H_1: Median of the difference NN-CL>0

P-value is **5.75×10^{-20}**, the null hypothesis is rejected.

H_0: Median of the difference TSP-CL=0

H_1: Median of the difference TSP-CL>0

P-value is ≈0; again there is no evidence to support the null hypothesis.

CL dominates the NN and TSP algorithms, gives better scheduling in the case of many tool changes.

iii) When die clearance changes dominate:

The hypotheses are the same as before. Testing the hypothesis NN-CL=0 versus NN-CL>0 yields a p-value of **0.99**; we fail to reject the null hypothesis. Testing the hypothesis TSP-CL=0 versus TSP-CL>0 results in a p-value of 0; so, there is no evidence to support the null hypothesis. The conclusion is that CL performs better than TSP, but there is not enough statistical evidence that it is better than NN. The findings of the sensitivity analyses are summarized in **Table 15**. In five out of six cases, CL yields better processing sequence when compared to others.

Table 15. Sensitivity analysis results.

	Tool Changes		Angle Changes		Die Clearance Changes	
Null hyp.	NN-CL=0	TSP-CL=0	NN-CL=0	TSP-CL=0	NN-CL=0	TSP-CL=0
P-value	**4.22×10^{-28}**	**0**	**5.75×10^{-20}**	**0**	**0.99**	**0**
Better alg.	CL	CL	CL	CL	Equal	CL

6. CONCLUSION

Setup time reduction is a challenge that many companies face during their daily operations. In this paper we analyzed cases where setup times are sequence-dependent and proposed several algorithms. We started with introducing the case company and the problem. Then, the methodologies that were used in this work were discussed. We received real data from a customer and also generated similar sample data using MATLAB to have better statistical reliability. Four different approaches, namely, Complete Enumeration, Nearest Neighbor, Clustering and Traveling Salesman Problem were employed to solve the problem under question. Complete Enumeration is the most optimal way to solve the problem but it is not feasible, because solution time increases exponentially with increasing number of jobs. TSP approach was used under several assumptions, but it did not give expected results and was dominated by CL and NN. Comparative analyses of CL and NN approaches showed that in majority of the cases, CL algorithm performs better. The sequence generated by this algorithm saved more than 10% of the setup times when compared to the default sequencing. At the end, sensitivity analysis was done to test the performance of the algorithms under various situations. The proposed algorithms solve such sequencing problems in less than 30 seconds on average.

The limitation of this study is that, we analyzed the data received from a customer of Prima Power Company which is operating in Finland. In the future, the researchers might attempt to replicate this work with more data from other companies. This would provide more information about the setup time savings of the proposed algorithms.

LIST OF REFERENCES

Adriaans, P., & Zantinge, D. Data mining, 1996. *Addision-Wesley, Harlow*.

Bowers, M. R., Groom, K., Ng, W. M., & Zhang, G. (1995). Cluster analysis to minimize sequence dependent changeover times. *Mathematical and computer modelling*, *21*(11), 89-95.

Burtseva, L., Yaurima, V. and Parra, R.R. (2010) *Scheduling Methods for Hybrid Flow Shops with Setup Times*, in *Handbook Future Manufacturing Systems*, India, InTech.

BusinessDictionary.com (2015). *Non value added activity*. [online] [Retrieved June 10, 2015]. Available from internet. URL: < http://www.businessdictionary.com/definition/non-value-added-activity.html>

Filho, M. G., de Fátima Morais, M., Boiko, T. J. P., Miyata, H. H., & Varolo, F. W. R. (2013). Scheduling in flow shop with sequence-dependent setup times: literature review and analysis. *International Journal of Business Innovation and Research*, *7*(4), 466-486.

Gupta, S. K. (1982). N jobs and m machines job-shop problems with sequence-dependent set-up times. *The International Journal of Production Research*, *20*(5), 643-656.

Reza Hejazi, S., & Saghafian, S. (2005). Flowshop-scheduling problems with makespan criterion: a review. *International Journal of Production Research*,*43*(14), 2895-2929.

Hwang, H., & Sun, J. U. (1998). Production sequencing problem with re-entrant work flows and sequence dependent setup times. *International Journal of Production Research*, *36*(9), 2435-2450.

Kenichi Sekine, Keisuke Arai (1998). TPM for the Lean Factory: Innovative Methods and Worksheets for Equipment Management. Productivity Press, Sep 24, 1998

Lockett, A. G., & Muhlemann, A. P. (1972). Technical Note—A Scheduling Problem Involving Sequence Dependent Changeover Times. *Operations Research*, *20*(4), 895-902.

Mirabi, M. (2011). Ant colony optimization technique for the sequence-dependent flowshop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, *55*(1-4), 317-326.

Nonaka, Y., Erdős, G., Kis, T., Nakano, T., & Váncza, J. (2012). Scheduling with alternative routings in CNC workshops. *CIRP Annals-Manufacturing Technology*, *61*(1), 449-454.

Papadimitriou, C. H. (1977). The Euclidean Traveling salesman problem is NP-complete. *Theoretical Computer Science*, *4*(3), 237-244.

Pinedo, M. (2008) *Scheduling: Theory, Algorithms, and Systems*, Prentice-Hall, New Jersey.

Spence, A. M., & Porteus, E. L. (1987). Setup reduction and increased effective capacity. *Management Science*, *33*(10), 1291-1301.

Valckenaers, P., & Van Brussel, H. (2005). Holonic manufacturing execution systems. *CIRP Annals-Manufacturing Technology*, *54*(1), 427-432.

White, C.H. & Wilson R.C. (1977). Sequence dependent setup times and job sequencing. *International Journal of Prod. Res.,* 15(2), 191-202.

Witten, I. H., & Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

Rao, Y., Huang, G., Li, P., Shao, X., & Yu, D. (2007). An integrated manufacturing information system for mass sheet metal cutting. *The International Journal of Advanced Manufacturing Technology*, *33*(5-6), 436-448.

von Luxburg, U., Bubeck, S., Jegelka, S., & Kaufmann, M. (1981). Nearest neighbor clustering. *Annals of Statistics*, *9*(1), 135-140.

Zandin, K. B. (Ed.). (2001). *Maynard's industrial engineering handbook*. New York, NY: McGraw-Hill.

APPENDIX 1. MATLAB code of the Complete Enumeration algorithm

```
%This code takes inputs about jobs, their requirements; tools, their
sizes;
%turret, its station sizes, etc. The output is the optimal (or
%near-optimal) sequencing of jobs.

function [ JobSequence, minimum_time_spent, last_position_of_turret  ]
= Brute_force_fn( number_of_tasks,
Test_job,initial_tools_in_turret1,turret_station_sizes,turret_station_
indexable,initial_tool_clearances,initial_tool_angles,tools,tool_sizex
,tool_change_time,clearance_change_time,angle_change_time,adap_plug_ti
me )

for i=1:number_of_tasks %5 or size(Test_job,2)
s(i).ToolNumber=Test_job(i).ToolRequirements(:,1)';
s(i).Angle=Test_job(i).ToolRequirements(:,2)';
s(i).Clearance=Test_job(i).ToolRequirements(:,3)';
end

%--*----*----*----*----*----*----*----*----*----*----*----*----*-
---*--
%This part does initializations of the varibales.

turret.ToolNumber=initial_tools_in_turret1;
turret.Turret_size=(hex2dec(turret_station_sizes')-9)';
turret.Clearance=initial_tool_clearances;
turret.Angle=initial_tool_angles;
tool_size=(hex2dec(tool_sizex')-9)';

initial_tools_in_turret=nonzeros(turret.ToolNumber);

initial=turret.ToolNumber;
cap=size(turret.ToolNumber,2);
turret1=struct;
cum_clearance_change=0;
clearance_change=0;
w=0;

%--*----*----*----*----*----*----*----*----*----*----*----*----*-
---*--
 %This part adds new column to "s" which is filled in with JobNumbers.

for r=1:size(s,2)
    s(r).JobNumber=r;
end
%--*----*----*----*----*----*----*----*----*----*----*----*----*-
---*--
```

```
p=perms(s);
adap=zeros(1,size(p,1));

for q=1:size(p,1)
    turret1(q,1).Angle=turret.Angle;
end


for q=1:size(p,1)
turret.ToolNumber=initial_tools_in_turret1;
turret.Clearance=initial_tool_clearances;
turret.Angle=initial_tool_angles;

for m=1:(size(s,2))


[~,~,dif]=find(setdiff(p(q,m).ToolNumber, turret.ToolNumber,
'stable')); %if size(dif,2)>0 then we need other options.
size_dif=size(dif,2);
turret_size_dif1=tool_size(dif) ;

union1=(nonzeros(union(turret.ToolNumber,p(q,m).ToolNumber,'stable')))
';

free_stations=find(turret.ToolNumber==0);

sizes_of_free_stations=turret.Turret_size(turret.ToolNumber==0);


if numel(dif)==0
    turret.ToolNumber;

elseif numel(union1)<=cap

for i=1:size_dif

turret_size_dif=tool_size(dif(i));
find_index_zeros=find(turret.ToolNumber==0);
a1=turret.Turret_size(find_index_zeros);  %Is there any free place
that is at the same size with dif?
a2=find(a1==turret_size_dif,1);
if numel(a2)==0
a2=find(a1>=turret_size_dif,1);
if (a1(a2)~=7 & turret_size_dif~=6)
 adap(q)=adap(q)+1;
end
end
if numel(a2)~=0
asda=find(turret.ToolNumber==0 & turret.Turret_size==a1(a2(1)) );
turret.ToolNumber(asda(1))=dif(i); %Here we add the extra tool to the
turret

end
end
```

```
tools_remaining_same=intersect(p(q,m).ToolNumber,turret.ToolNumber,
'stable'); %This part assigns the clearances. Same logic applies for
angles.
for i=1:numel(tools_remaining_same)
[~,index3(i),~]=find(turret.ToolNumber==tools_remaining_same(i));
end
for j=1:numel(tools_remaining_same)
  [~,index4(j),~]=find(p(q,m).ToolNumber==tools_remaining_same(j));
end
for a1=1:numel(tools_remaining_same)
 turret.Clearance(index3(a1))= p(q,m).Clearance(index4(a1));
end


for a1=1:numel(tools_remaining_same)
    if abs( turret.Angle(index3(a1))- p(q,m).Angle(index4(a1)))~=180
    turret.Angle(index3(a1))= p(q,m).Angle(index4(a1));
    end
end

if m==1
tools_remaining_same=intersect(turret.ToolNumber,p(q,m).ToolNumber,
'stable');
else
tools_remaining_same=intersect(turret1(q,m-
1).ToolNumber,p(q,m).ToolNumber, 'stable');
end
for i=1:numel(tools_remaining_same)
    if m==1
[~,index3(i),~]=find(turret.ToolNumber==tools_remaining_same(i));
    else
        [~,index3(i),~]=find(turret1(q,m-
1).ToolNumber==tools_remaining_same(i));
    end
end
for j=1:numel(tools_remaining_same)
  [~,index4(j),~]=find(p(q,m).ToolNumber==tools_remaining_same(j));
end


for a1=1:numel(tools_remaining_same)
    if m==1
        if abs(turret1(q,m).Angle(index3(a1))-
p(q,m).Angle(index4(a1)))~=180
            turret.Angle(index3(a1))= p(q,m).Angle(index4(a1));
        end
    elseif abs(turret1(q,m-1).Angle(index3(a1))-
p(q,m).Angle(index4(a1)))~=180
    turret.Angle(index3(a1))= p(q,m).Angle(index4(a1));  %This part
assigns the angles.
    end
end

else

for b=dif
```

```
    af=find(turret.ToolNumber==0)  ;
    for a=af     %This part fills in the turrets with the tools with
equal size.

        if turret.Turret_size(a)==tool_size(b)
            turret.ToolNumber(a)=b; %Here we add the extra tool to the
turret
            break
        end

    end

end
[~,~,dif]=find(setdiff(p(q,m).ToolNumber, turret.ToolNumber,
'stable'));
turret_size_dif1=tool_size(dif) ;


as=numel(dif);
for j=1:as
    free_stations=find(turret.ToolNumber==0);
for i=free_stations                              %This part fills in
the tools into larger stations (for the ones that were not placed at
the top).

    if turret.Turret_size(i)>=tool_size(dif(j))
        turret.ToolNumber(i)=dif(j);
        if (turret.Turret_size(i)~=7 & tool_size(dif(j))~=6)
       adap(q)=adap(q)+1;
        end
       break
    end
end



end

[~,~,dif]=find(setdiff(p(q,m).ToolNumber, turret.ToolNumber,
'stable'));

% Up to here:
% 1. The difference is found.
% 2. The ones that have the same size of free stations, are placed.
% 3. The ones that can fit to bigger stations are placed.
% 4. Now, the turret is full. We have to remove the unneeded tools to
get
% all the tools ready in turret.

for c=dif
    Unneeded_stations=0;
    Unneeded_tools=0;

[~,~,Unneeded_tools]=find(setdiff(turret.ToolNumber,p(q,m).ToolNumber,
'stable'));
```

```
for h=1:numel(Unneeded_tools)
Unneeded_stations(h)=find(turret.ToolNumber==Unneeded_tools(h));
end

    for d=Unneeded_stations
       if tool_size(c)==turret.Turret_size(d) %This part finds the
same sized stations and places the tool.
            turret.ToolNumber(d)=c;
            break
       end

    end

end



[~,~,dif]=find(setdiff(p(q,m).ToolNumber, turret.ToolNumber,
'stable'));
[~,~,Unneeded_tools]=find(setdiff(turret.ToolNumber,p(q,m).ToolNumber,
'stable'));


for h=1:numel(Unneeded_tools)
Unneeded_stations(h)=find(turret.ToolNumber==Unneeded_tools(h));
end

if numel(dif)~=0

for e=dif

[~,~,Unneeded_tools]=find(setdiff(turret.ToolNumber,p(q,m).ToolNumber,
'stable'));

for h=1:numel(Unneeded_tools)
Unneeded_stations(h)=find(turret.ToolNumber==Unneeded_tools(h));
end
    for f=Unneeded_stations

sizes_of_unneeded_stations=turret.Turret_size(Unneeded_stations);
        if tool_size(e)<=turret.Turret_size(f)
            turret.ToolNumber(f)=e;
             if (turret.Turret_size(f)~=7 & tool_size(e)~=6)%This part
assigns a tool to larger station (adapter is used in this case).
            adap(q)=adap(q)+1;
             end
             break
        end

    end
end

end

end
```

```
tools_remaining_same=intersect(p(q,m).ToolNumber,turret.ToolNumber,
'stable');
for i=1:numel(tools_remaining_same)
[~,index3(i),~]=find(turret.ToolNumber==tools_remaining_same(i));
end
for j=1:numel(tools_remaining_same)
  [~,index4(j),~]=find(p(q,m).ToolNumber==tools_remaining_same(j));
end
for a1=1:numel(tools_remaining_same)
 turret.Clearance(index3(a1))= p(q,m).Clearance(index4(a1));
end

for a1=1:numel(tools_remaining_same)
    if abs( turret.Angle(index3(a1))- p(q,m).Angle(index4(a1)))~=180
    turret.Angle(index3(a1))= p(q,m).Angle(index4(a1));
    end
end

if m==1
tools_remaining_same=intersect(turret.ToolNumber,p(q,m).ToolNumber,
'stable');
else
tools_remaining_same=intersect(turret1(q,m-
1).ToolNumber,p(q,m).ToolNumber, 'stable');
end
for i=1:numel(tools_remaining_same)
    if m==1
[~,index3(i),~]=find(turret.ToolNumber==tools_remaining_same(i));
    else
        [~,index3(i),~]=find(turret1(q,m-
1).ToolNumber==tools_remaining_same(i));
    end
end

for j=1:numel(tools_remaining_same)
  [~,index4(j),~]=find(p(q,m).ToolNumber==tools_remaining_same(j));
end

for a1=1:numel(tools_remaining_same)
    if m==1
        if abs(turret1(q,m).Angle(index3(a1))-
p(q,m).Angle(index4(a1)))~=180
            turret.Angle(index3(a1))= p(q,m).Angle(index4(a1));
        end
    elseif abs(turret1(q,m-1).Angle(index3(a1))-
p(q,m).Angle(index4(a1)))~=180
    turret.Angle(index3(a1))= p(q,m).Angle(index4(a1));  %This part
assigns the angles.
    end
end

    turret1(q,m).ToolNumber=turret.ToolNumber;
    turret1(q,m).Clearance=turret.Clearance;
    turret1(q,m).Angle=turret.Angle;
```

```
end

end

for q=1:size(turret1,1)
    w(1)=size(setdiff(turret1(q,1).ToolNumber,initial),2);
    for m=2:size(turret1,2)
          w(m)=size(setdiff(turret1(q,m).ToolNumber,turret1(q,m-
1).ToolNumber),2);
    end
    total_change(q)=sum(w);
end

%This part does clearance calculations

for q=1:size(turret1,1)
    clearance_change=0;
    for m=1:size(turret1,2)
        if m==1
           x=nonzeros(intersect(turret1(q,m).ToolNumber,initial,
'stable'));
           for ad=1:numel(x)
            [~,index5(ad),~]=find(turret1(q,m).ToolNumber==x(ad));
            [~,index6(ad),~]=find(initial==x(ad));
            if
turret1(q,m).Clearance(index5(ad))~=initial_tool_clearances(index6(ad)
)
                clearance_change=clearance_change+1;
            end
           end
        else
            x=nonzeros(intersect(turret1(q,m).ToolNumber,turret1(q,m-
1).ToolNumber, 'stable'));
            for ad=1:numel(x)
            [~,index5(ad),~]=find(turret1(q,m).ToolNumber==x(ad));
            [~,index6(ad),~]=find(turret1(q,m-1).ToolNumber==x(ad));
            if turret1(q,m).Clearance(index5(ad))~=turret1(q,m-
1).Clearance(index6(ad))
                clearance_change=clearance_change+1;
            end
           end
        end
    end
   cum_clearance_change(q)=clearance_change;
end

clear index5 index6 x

for q=1:size(turret1,1)
   angle_change=0;
    for m=1:size(turret1,2)
        if m==1
           x=nonzeros(intersect(turret1(q,m).ToolNumber,initial,
'stable'));
           for ad=1:numel(x)
            [~,index5(ad),~]=find(turret1(q,m).ToolNumber==x(ad));
```

```
            [~,index6(ad),~]=find(initial==x(ad));
            if
(turret1(q,m).Angle(index5(ad))~=initial_tool_angles(index6(ad)) &
turret_station_indexable(index5(ad))==0)
                angle_change=angle_change+1;
            end
           end
        else
            x=nonzeros(intersect(turret1(q,m).ToolNumber,turret1(q,m-
1).ToolNumber, 'stable'));
            for ad=1:numel(x)
            [~,index5(ad),~]=find(turret1(q,m).ToolNumber==x(ad));
            [~,index6(ad),~]=find(turret1(q,m-1).ToolNumber==x(ad));
            if (turret1(q,m).Angle(index5(ad))~=turret1(q,m-
1).Angle(index6(ad)) & turret_station_indexable(index5(ad))==0)
                angle_change=angle_change+1;
            end
           end
        end
    end
   cum_angle_change(q)=angle_change;

end

for r=1:size(p,1)
    total_time(r)=total_change(r)*tool_change_time +
cum_angle_change(r)*angle_change_time
+cum_clearance_change(r)*clearance_change_time+adap(r)*adap_plug_time
; % This part calculates total time spent for all combinations.
end

[~,index_of_min,~]=find(total_time==min(total_time),1);


for n=1:size(s,2)
    JobSequence(n)=p(index_of_min,n).JobNumber;
end

turret1(index_of_min,:).ToolNumber;
minimum_time_spent=min(total_time);
JobSequence;
last_position_of_turret=turret1(index_of_min,number_of_tasks);
clearvars -except times s adap turret1 JobSequence
last_position_of_turret minimum_time_spent p total_time index_of_min
total_change cum_angle_change cum_clearance_change

end
```

APPENDIX 2. MATLAB code of the Nearest Neighbor algorithm

```
function [ visited, total_time] = NN_fn(
Test_job,initial_tools_in_turret1,turret_station_sizes,turret_station_
indexable,initial_tool_clearances,initial_tool_angles,tools,tool_sizex
,tool_change_time,clearance_change_time,angle_change_time,adap_plug_ti
me )
%This code takes inputs about jobs, their requirements; tools, their
sizes;
%turret, its station sizes, etc. The output is the optimal (or
%near-optimal) sequencing of jobs.

for i=1:size(Test_job,2)
s(i).ToolNumber=Test_job(i).ToolRequirements(:,1)';
s(i).Angle=Test_job(i).ToolRequirements(:,2)';
s(i).Clearance=Test_job(i).ToolRequirements(:,3)';
end

turret.ToolNumber=initial_tools_in_turret1;
turret.Turret_size=(hex2dec(turret_station_sizes')-9)';
turret.Clearance=initial_tool_clearances;
turret.Angle=initial_tool_angles;
tool_size=(hex2dec(tool_sizex')-9)';

initial=turret.ToolNumber;
cap=size(turret.ToolNumber,2);
turret1=struct;
turret2=struct;
cum_clearance_change=0;
cum_angle_change=0;
clearance_change=0;



for r=1:size(s,2)
    s(r).JobNumber=r; %This part adds new column to "s" which is
filled in with JobNumbers.
end
 turret1(1).Angle=turret.Angle;
 turret1(1).ToolNumber=turret.ToolNumber;
 turret1(1).Clearance=turret.Clearance;
adap=0;
adap1=0;
unvisited=[1:size(s,2)];
visited=[];
hj=1;
previous_changes=0;
for q=2:(size(s,2)+1)

ij=1;

total_change=[];
```

```
total_time=[];

    for m=unvisited
 adap(q,m)=adap1;
 clearance_change(q,m)=cum_clearance_change;
 angle_change(q,m)=cum_angle_change;
turret2(q,m).ToolNumber=turret1(q-1).ToolNumber;
turret2(q,m).Clearance=turret1(q-1).Clearance;
turret2(q,m).Angle=turret1(q-1).Angle;
[~,~,dif]=find(setdiff(s(m).ToolNumber, turret1(q-1).ToolNumber,
'stable')); %if size(dif,2)>0 then we need other options.
size_dif=size(dif,2);
turret_size_dif1=tool_size(dif) ;

union1=(nonzeros(union(turret1(q-
1).ToolNumber,s(m).ToolNumber,'stable')))';

free_stations=find(turret1(q-1).ToolNumber==0);
sizes_of_free_stations=turret.Turret_size(free_stations);


if numel(dif)==0
    turret1(q-1).ToolNumber;

elseif numel(union1)<=cap

for i=1:size_dif
turret_size_dif=tool_size(dif(i));
find_index_zeros=find(turret2(q,m).ToolNumber==0);
a1=turret.Turret_size(find_index_zeros); %Is there any free place that
is at the same size with dif?
a2=find(a1==turret_size_dif,1);
if numel(a2)==0
a2=find(a1>=turret_size_dif,1);
if (a1(a2)~=7 & turret_size_dif~=6)
 adap(q,m)=adap(q,m)+1;
end
end
if numel(a2)~=0
 asda=find(turret2(q,m).ToolNumber==0 & turret.Turret_size==a1(a2(1))
);
turret2(q,m).ToolNumber(asda(1))=dif(i); %Here we add the extra tool
to the turret
end
end
tools_remaining_same=unique(intersect(turret2(q,m).ToolNumber,s(m).Too
lNumber)); %This part assigns the clearances. Same logic applies for
angles.
for i=1:numel(tools_remaining_same)
[~,index3(i),~]=find(unique(s(m).ToolNumber==tools_remaining_same(i)))
;
end

for j=1:numel(tools_remaining_same)
```

```
[~,index4(j),~]=find(turret2(q,m).ToolNumber==tools_remaining_same(j))
;
end
for a1=1:numel(tools_remaining_same)
 turret2(q,m).Clearance(index4(a1))=s(m).Clearance(index3(a1));
end

for a1=1:numel(tools_remaining_same)
    if abs(s(m).Angle(index3(a1))-
turret2(q,m).Angle(index4(a1)))~=180
    turret2(q,m).Angle(index4(a1))= s(m).Angle(index3(a1));  % This
should be done when the difference is not 180 degrees.
    end
end


tools_remaining_same=intersect(turret2(q,m).ToolNumber,s(m).ToolNumber
, 'stable');

for i=1:numel(tools_remaining_same)
[~,index3(i),~]=find(turret2(q,m).ToolNumber==tools_remaining_same(i))
;
end

for j=1:numel(tools_remaining_same)

[~,index4(j),~]=find(unique(s(m).ToolNumber==tools_remaining_same(j)))
;
end


for a1=1:numel(tools_remaining_same)

    turret2(q,m).Angle(index3(a1))= s(m).Angle(index4(a1));  %This
part assigns the angles. But make sure that you consider the opposite
angles issue in your calculations.

end

else

for b=dif
    af=find(turret2(q,m).ToolNumber==0)  ;
    for a=af     %This part fills in the turrets with the tools with
equal size.

        if turret.Turret_size(a)==tool_size(b)
            turret2(q,m).ToolNumber(a)=b;
            break
        end

    end

end
```

```
[~,~,dif]=find(setdiff(s(m).ToolNumber, turret2(q,m).ToolNumber,
'stable'));
turret_size_dif1=tool_size(dif) ;


as=numel(dif);
for j=1:as
    free_stations=find(turret2(q,m).ToolNumber==0);
for i=free_stations                              %This part fills in
the tools into larger stations (for the ones that were not placed at
the top).

    if turret.Turret_size(i)>=tool_size(dif(j))
        turret2(q,m).ToolNumber(i)=dif(j);
        if (turret.Turret_size(i)~=7 & tool_size(dif(j))~=6)
       adap(q,m)=adap(q,m)+1;
        end
          break
    end
end


end

[~,~,dif]=find(setdiff(s(m).ToolNumber, turret2(q,m).ToolNumber,
'stable'));

% Up to here:
% 1. The difference is found.
% 2. The ones that have the same size of free stations, are placed.
% 3. The ones that can fit to bigger stations are placed.
% 4. Now, the turret is full. We have to remove the unneeded tools to
get
% all the tools ready in turret.

for c=dif
    Unneeded_stations=0;
    Unneeded_tools=0;

[~,~,Unneeded_tools]=find(setdiff(turret2(q,m).ToolNumber,s(m).ToolNum
ber, 'stable'));

for h=1:numel(Unneeded_tools)
Unneeded_stations(h)=find(turret2(q,m).ToolNumber==Unneeded_tools(h));
end

    for d=Unneeded_stations
       if tool_size(c)==turret.Turret_size(d) %This part finds the
same sized stations and places the tool.
            turret2(q,m).ToolNumber(d)=c;
            break %I ADDED NOW!!!
       end

    end
```

```
end



[~,~,dif]=find(setdiff(s(m).ToolNumber, turret2(q,m).ToolNumber,
'stable'));
[~,~,Unneeded_tools]=find(setdiff(turret2(q,m).ToolNumber,s(m).ToolNum
ber, 'stable'));



for h=1:numel(Unneeded_tools)
Unneeded_stations(h)=find(turret2(q,m).ToolNumber==Unneeded_tools(h));
end

if numel(dif)~=0

for e=dif

[~,~,Unneeded_tools]=find(setdiff(turret2(q,m).ToolNumber,s(m).ToolNum
ber, 'stable'));

for h=1:numel(Unneeded_tools)
Unneeded_stations(h)=find(turret2(q,m).ToolNumber==Unneeded_tools(h));
end
    for f=Unneeded_stations

sizes_of_unneeded_stations=turret.Turret_size(Unneeded_stations);
        if tool_size(e)<=turret.Turret_size(f)
            turret2(q,m).ToolNumber(f)=e;
             if (turret.Turret_size(f)~=7 & tool_size(e)~=6)%This part
assigns a tool to larger station (adapter is used in this case).
            adap(q,m)=adap(q,m)+1;
             end
             break
        end

    end
end

end

end



tools_remaining_same=intersect(turret2(q,m).ToolNumber,s(m).ToolNumber
, 'stable'); %This part assigns the clearances. Same logic applies for
angles.
for i=1:numel(tools_remaining_same)
[~,index3(i),~]=find(unique(s(m).ToolNumber==tools_remaining_same(i)))
;
end
for j=1:numel(tools_remaining_same)
```

```
[~,index4(j),~]=find(turret2(q,m).ToolNumber==tools_remaining_same(j))
;
end
for a1=1:numel(tools_remaining_same)
 turret2(q,m).Clearance(index4(a1))=s(m).Clearance(index3(a1));
end



%This part assigns angles.


for a1=1:numel(tools_remaining_same)
    if abs(s(m).Angle(index3(a1))-
turret2(q,m).Angle(index4(a1)))~=180
    turret2(q,m).Angle(index4(a1))= s(m).Angle(index3(a1));  % This
should be done when the difference is not 180 degrees.
    end
end


tools_remaining_same=intersect(turret2(q,m).ToolNumber,s(m).ToolNumber
, 'stable');

for i=1:numel(tools_remaining_same)
[~,index3(i),~]=find(turret2(q,m).ToolNumber==tools_remaining_same(i))
;
end

for j=1:numel(tools_remaining_same)

[~,index4(j),~]=find(unique(s(m).ToolNumber==tools_remaining_same(j)))
;
end


for a1=1:numel(tools_remaining_same)

    turret2(q,m).Angle(index3(a1))= s(m).Angle(index4(a1));  %This
part assigns the angles. But make sure that you consider the opposite
angles issue in your calculations.

end


%This part does clearance calculations
         x=nonzeros(intersect(turret2(q,m).ToolNumber,turret1(q-
1).ToolNumber, 'stable'));
            for ad=1:numel(x)
            [~,index5(ad),~]=find(turret2(q,m).ToolNumber==x(ad));
            [~,index6(ad),~]=find(turret1(q-1).ToolNumber==x(ad));
            if turret2(q,m).Clearance(index5(ad))~=turret1(q-
1).Clearance(index6(ad))
                clearance_change(q,m)=clearance_change(q,m)+1;
```

```
            end
           end


%This part does angle calculations.



clear index5 index6 x

            x=nonzeros(intersect(turret2(q,m).ToolNumber,turret1(q-
1).ToolNumber, 'stable'));
            for ad=1:numel(x)
            [~,index5(ad),~]=find(turret2(q,m).ToolNumber==x(ad));
            [~,index6(ad),~]=find(turret1(q-1).ToolNumber==x(ad));
            if (turret2(q,m).Angle(index5(ad))~=turret1(q-
1).Angle(index6(ad)) & turret_station_indexable(index5(ad))==0)
                angle_change(q,m)=angle_change(q,m)+1;
            end
           end
total_change(ij)=previous_changes+size(setdiff(turret2(q,m).ToolNumber
,turret1(q-1).ToolNumber),2);
total_time(ij)=total_change(ij)*tool_change_time+adap(q,m)*adap_plug_t
ime+clearance_change(q,m)*clearance_change_time+angle_change(q,m)*angl
e_change_time;
ij=ij+1;
    end
    [~,index_of_min,~]=find(total_time==min(total_time(:)),1);
    visited(hj)=unvisited(index_of_min);
    hj=hj+1;

turret1(q).ToolNumber=turret2(q,unvisited(index_of_min)).ToolNumber;
    turret1(q).Clearance=turret2(q,unvisited(index_of_min)).Clearance;
    turret1(q).Angle=turret2(q,unvisited(index_of_min)).Angle;
        if numel(total_change)==0
        total_change(1)=0;
    end

    previous_changes=total_change(index_of_min);
    adap1=adap(q,unvisited(index_of_min));
    cum_clearance_change=clearance_change(q,unvisited(index_of_min));
    cum_angle_change=angle_change(q,unvisited(index_of_min));
    index_of_min;
    unvisited(index_of_min)=[];

  end
visited;

total_time;

end
```

APPENDIX 3. MATLAB code of the Clustering algorithm

```
function [ visited_sequencex, total_time ] =
Cluster_fn2(Test_job,initial_tools_in_turret1, turret_station_sizes,
turret_station_indexable, initial_tool_clearances,
initial_tool_angles, tools, tool_sizex,
tool_change_time,clearance_change_time,angle_change_time,adap_plug_tim
e)

unvisited=[1:size(Test_job,2)];

ToolNumber=struct;
ToolNumber(1).a=nonzeros(initial_tools_in_turret1)';

for p=2:size(Test_job,2)+1
    ToolNumber(p).a=Test_job(p-1).ToolRequirements(:,1)';
end

distance_matrix = zeros(size(ToolNumber,2),size(ToolNumber,2));

last_position_of_turret.ToolNumber=initial_tools_in_turret1;
last_position_of_turret.Clearance=initial_tool_clearances;
last_position_of_turret.Angle=initial_tool_angles;

for i=1:size(ToolNumber,2)
for j=1:size(ToolNumber,2)

distance_matrix(i,j)=size(setdiff(ToolNumber(i).a,ToolNumber(j).a),2);
    if i==j
    distance_matrix(i,j)=100;
    end
end
end

for i=2:size(ToolNumber,2)
   distance_matrix(i,1)=1000;
end


number_of_tasks=5;
no_of_jobs=size(Test_job,2);
start=1;
visited_sequencex=[];
no_of_full_clusters = floor(no_of_jobs/5);
size_of_extra=no_of_jobs-no_of_full_clusters*5;

for u=1:no_of_full_clusters
    j=1;
unsorted=distance_matrix(start,:); % We take out the row of the
starting point.
```

```
%unsorted(start)=[];        %We take out the cell corresponding to the
(n,n)th element of the distance matrix because it is zero by default.
This ensures that we don't go from one point to itself.
sorted=sort(distance_matrix(start,:)); %We sort the vector.
%sorted(start)=[];
top_distances=sorted([1:5]); %The first 5 shortest distances are taken
out.
top_tasks=[]; %Initialization.

for i=1:5
as=find(unsorted==top_distances(i));
top_tasks=unique(horzcat(top_tasks,as),'stable');
end
top5_tasks=top_tasks(1:5)-1;

for i=1:5

Test_job1(i).ToolRequirements=Test_job(top5_tasks(i)).ToolRequirements
;
end

initial_tools_in_turret1=last_position_of_turret.ToolNumber;
initial_tool_clearances=last_position_of_turret.Clearance;
initial_tool_angles=last_position_of_turret.Angle;
[sub_sequence, times(u), last_position_of_turret ]
=Brute_force_fn(number_of_tasks,Test_job1,initial_tools_in_turret1,tur
ret_station_sizes,turret_station_indexable,initial_tool_clearances,ini
tial_tool_angles,tools,tool_sizex,tool_change_time,clearance_change_ti
me,angle_change_time,adap_plug_time );

visited_sequence=[];

for i=sub_sequence
visited_sequence(j)=top5_tasks(i);
distance_matrix(:,visited_sequence(j)+1)=[100];
j=j+1;
end
start=visited_sequence(end)+1;
visited_sequencex=horzcat(visited_sequencex,visited_sequence);

end

if size_of_extra~=0
u=no_of_full_clusters+1;
j=1;
unsorted=distance_matrix(start,:);
%unsorted(start)=[];
sorted=sort(distance_matrix(start,:));
%sorted(start)=[];
top_distances=sorted([1:size_of_extra]);
top_tasks=[];

for i=1:size_of_extra
 as=find(unsorted==top_distances(i));
top_tasks=unique(horzcat(top_tasks,as),'stable');
end
```

```
top5_tasks=top_tasks(1:size_of_extra)-1;


for i=1:size_of_extra

Test_job1(i).ToolRequirements=Test_job(top5_tasks(i)).ToolRequirements
;
end
initial_tools_in_turret1=last_position_of_turret.ToolNumber;
initial_tool_clearances=last_position_of_turret.Clearance;
initial_tool_angles=last_position_of_turret.Angle;
number_of_tasks=numel(top5_tasks);
[sub_sequence,
times(u)]=Brute_force_fn(number_of_tasks,Test_job1,initial_tools_in_tu
rret1,turret_station_sizes,turret_station_indexable,initial_tool_clear
ances,initial_tool_angles,tools,tool_sizex,tool_change_time,clearance_
change_time,angle_change_time,adap_plug_time );
visited_sequence=[];
for i=sub_sequence
visited_sequence(j)=top5_tasks(i);
distance_matrix(:,visited_sequence(j)+1)=[100];
j=j+1;
end
start=visited_sequence(end)+1;
visited_sequencex=horzcat(visited_sequencex,visited_sequence);
end

unvisited(visited_sequencex)=[];
visited_sequencex;
unvisited;
total_time=sum(times);

end
```

APPENDIX 4. MATLAB code of the Traveling Salesman Problem algorithm

```
function [optimal_route,number_of_changes] = TSP_fn(Test_job)
%This function uses Traveling Salesman Approach to solve sequencing
%problem.
ToolNumber=struct;
for p=2:size(Test_job,2)+1
    ToolNumber(p).a=Test_job(p-1).ToolRequirements(:,1)';
end

distance_matrix = zeros(size(ToolNumber,2),size(ToolNumber,2));

for i=1:size(ToolNumber,2)
for j=1:size(ToolNumber,2)

distance_matrix(i,j)=size(setdiff(ToolNumber(i).a,ToolNumber(j).a),2);
end
end
for i=2:size(ToolNumber,2)
   distance_matrix(i,1)=1000;
end
userConfig.xy=rand(size(ToolNumber,2),2);
userConfig.dmat=distance_matrix;
userConfig.showProg=0;
userConfig.showResult=0;
userConfig.numIter=5000;
resultStruct = tspo_ga(userConfig);
optimal_route=nonzeros(resultStruct.optRoute-1)';
%(resultStruct.optRoute-1)
number_of_changes=resultStruct.minDist;
%per=perms([1:(size(ToolNumber,2)-1)]);
%aw=find(per(:,1)==4 & per(:,2)==1 & per(:,3)==2 & per(:,4)==3  &
per(:,5)==5);
%total_time_spent=total_time(aw)
end
```