DISSERTATION

METAHEURISTIC APPROACH TO SOLVING U-SHAPED ASSEMBLY LINE BALANCING PROBLEMS USING A RULE-BASE CODED GENETIC ALGORITHM

Submitted by

Ulises Martinez-Contreras

Department of Mechanical Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2015

ABSTRACT

METAHEURISTIC APPROACH TO SOLVING U-SHAPED ASSEMBLY LINE BALANCING PROBLEMS USING A RULE-BASE CODED GENETIC ALGORITHM

The need to achieve line balancing for a U-shaped production line to minimize production time and cost is a problem frequently encountered in industry. This research presents an efficient and quick algorithm to solve the U-shape line-balancing problem. Heuristic rules used to solve a straight line-balancing problem (LBP) were modified and adapted so they could be applied in a U-shape line-balancing problem model. By themselves, the heuristic rules, which were adapted from straight-line systems, can produce good solutions for the U-shape LBP, however, there is nothing that guarantees that this will be the case. One way to achieve improved solutions using heuristic rules can be accomplished by using a number of rules simultaneously to break ties during the task assignment process. In addition to the use of heuristic and simultaneous heuristic rules, basic genetic operations were used to further improve the performance of the assignment process and thus obtain better solutions.

Two genetic algorithms are introduced in this research: a direct-coded and an indirect-coded model. The newly introduced algorithms were compared with well-known problems from literature and their performance as compared to other heuristic approaches showed that they perform well.

The indirect-coded genetic algorithm uses the adapted heuristic rules from the LBP as genes to find the solutions to the problem. In the direct-coded algorithm, each gene represents an operation in the LBP and the position of the gene in the chromosome represents the order in which an operation, or task, will be assigned to a workstation.

The indirect-coded genetic algorithm introduces sixteen heuristic rules adapted from the straight LBP for use in a U-shape LBP. Each heuristic rule was represented inside the chromosome as a gene. The rules were implemented in a way that precedence is preserved and at the same time, facilitate the use of genetic operations. Comparing the algorithm's results with known results from literature, it obtained better solutions in 26% of the cases; it obtained an equivalent solution in 62% of the cases (not better, not worse); and a worse solution the remaining 12%.

The direct-coded genetic algorithm introduces a new way to construct an ordered arrangement of the task assignation without violating any precedence. This method consists of creating a diagram that is isomorphic to the original precedence diagram to facilitate the construction of the chromosome. Also, crossover and mutation operations are conducted in a way that precedence relations are not violated. The direct-coded genetic algorithm was tested with the same set of problems as the indirect-coded algorithm. It obtained better solutions than the known solutions from literature in 22% of the cases; 72% of the problems had an equivalent solution; and 6% of the time it generated a solution less successful than the solution from literature.

Something that had not been used in other genetic algorithm studies is a response surface methodology to optimize the levels for the parameters that are involved in the response model. The response surface methodology is used to find the best values for the parameters (% of children, % of mutations, number of genes, number of chromosomes) to produce good solutions for problems of different sizes (large, medium, small). This allows for the best solution to be obtained in a minimum amount of time, thus saving computational effort.

Even though both algorithms produce good solutions, the direct-coded genetic algorithm option requires less computational effort. Knowing the capabilities of genetic algorithms, they were then tested in two real industry problems to improve assembly-line functions. This resulted in increased efficiency in both production line.

ACKNOWLEDGMENTS

I would like to express my sincerest gratitude to my advisor and mentor, Dr. William S. Duff. I am blessed to have had the opportunity to work with him and learn from him in the past years. He gave me guidance, constant encouragement, constructive criticism, and enormous patience throughout all these years.

I want to thank my wife, Mirella Parada-Gonzalez, for her love, understanding, patience, and encouragement, to my son Ulises Martinez-Parada for being a very important part of this accomplishment, to my son Hector Martinez-Parada and my daughter Mirella Martinez-Parada. My family has been encouraging, supportive and shown belief in me and my work. I could not have accomplished this without them.

Unfortunately some people who had great influence on me are no longer with us, I am very grateful to my parents Erasmo and Jovita for their constant and unconditional love and support. My brother Salvador who was always a great support for me at different stages of my life.

Finally, I would like to thank my brother, my sisters and all the people who one way or another have been a part of my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Nowadays, companies around the world are producing high-quality products to sell at the lowest possible price. This is not because they do not want to raise more revenue through the sale of products but rather, they are facing the necessity of increasing their participation in the market because competitors also are selling high-quality products at low prices. There are several techniques to continuously improve quality and reduce operation costs. One of these techniques is called Line Balancing. The line balancing problem (LBP) entails assigning approximately the same amount of workload to each workstation, or worker, in an assembly line. According to Milas (1990), "The assembly line exists when we assemble or handle any device or product in a planned, sequential manner with two or more operators performing tasks of repetitive work at established workstations."

Assembly line configurations are among the most important components in manufacturing. Almost all industrial problems come from some sort of assembly line. The most fundamental obstacle to overcome to gain efficiency in the assembly line configuration is to balance the line (i.e., to solve the LBP). When the product has many operations and the demand is high, the process of balancing the line becomes more and more difficult. According to Ajenblit (1998), there are two types of optimization problems for the LBP. "In the Type I problem, the cycle-time (maximum amount of time that can be spent at each workstation) is fixed and the objective is to minimize the required number of workstations. The Type II problem attempts to minimize the maximum cycle-time, given a fixed number of workstations." Type II balancing problems generally occur when the organization wants to produce the optimum number of items using a fixed number of stations without any expansion. (Li-yun X. et al. 2014). In industry, the Type I LBP is most prevalent, thus this research is focused only on the Type I LBP.

Just-in-Time (JIT) and Lean Manufacturing methods have driven industries to search for manufacturing methodologies with lean concepts and high efficiency in both inventory and labor. This is why many assembly lines are now being designed U-shaped. A straight production line is shown in Figure 1.1.

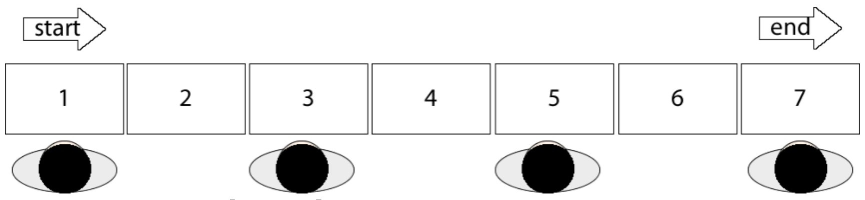


Figure 1.1: Straight Production Line

Assembly line workstations arranged in a line are not the most efficient arrangement. In a U-shaped line balancing arrangement the workers operate inside the U, which offers some advantages that the straight-line shape assembly does not: communication among workers is better, workers can see other processes, and it is easier to assign multiple workstations to a single operator. When in-line production systems are compared with U-shaped production systems, the latter always assigns a more balanced amount of work to the operators. This always results in fewer workers and better material handling. Figure 1.2 shows the flow of the U-shaped assembly line.

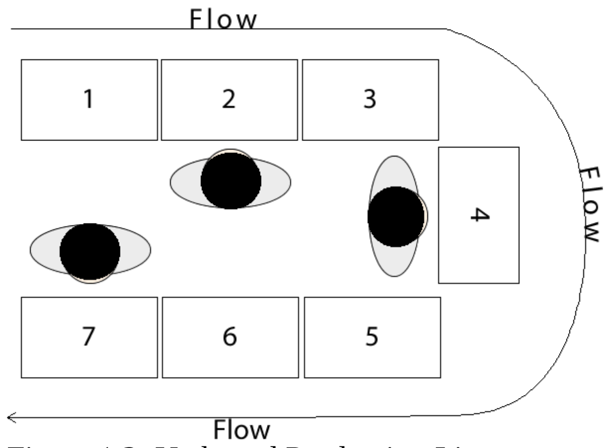


Figure 1.2: U-shaped Production Line

The LBP offers manufacturers the opportunity to reduce the number of operators through optimizing the process of assigning tasks. This is why the LBP is of great interest among researchers and many methods have been developed to find a solution. However, it was not until 1994 that the first research work was published regarding the U-shaped balancing problem (Miltenburg and Wijngaard, 1994).

Miltenburg and Wijngaard (1994) adopted the heuristic methods developed for the traditional LBP and adapted them for use in the U-shaped form. They came up with a procedure involving dynamic programming, but due to the high computational costs of this technique, it was used only for small problems.

This research introduces two heuristic methods, called Genetic Algorithms, using direct- and indirect-codification. These algorithms combine a rule-based method with an evolutionary algorithm. The aforementioned method will help in solving the Type I U-shaped balancing problem, providing solutions that simultaneously satisfy the precedence restrictions, result in fewer work stations, minimize the smoothness index (Formula [1]), and do not exceed a given cycle time.

$$f(x) = minimize \sqrt{\sum_{k=1}^{m} \frac{(s_{max} - s_k)^2}{m}} \qquad [1]$$

where s_k is the total workstation k time, s_{max} is the total maximum time of all workstations, and m is the number of workstations.

The inputs for design of the assembly line system are listed below:

- Precedence network of tasks,
- Task times, which may be either deterministic or probabilistic, and
- Cycle time, or number of workstations.

The precedence network defines the immediate precedence relationships among the tasks of assembling a product. Figure 1.3 shows a typical precedence diagram where each circle represents an operation and the number above it indicates its task time.

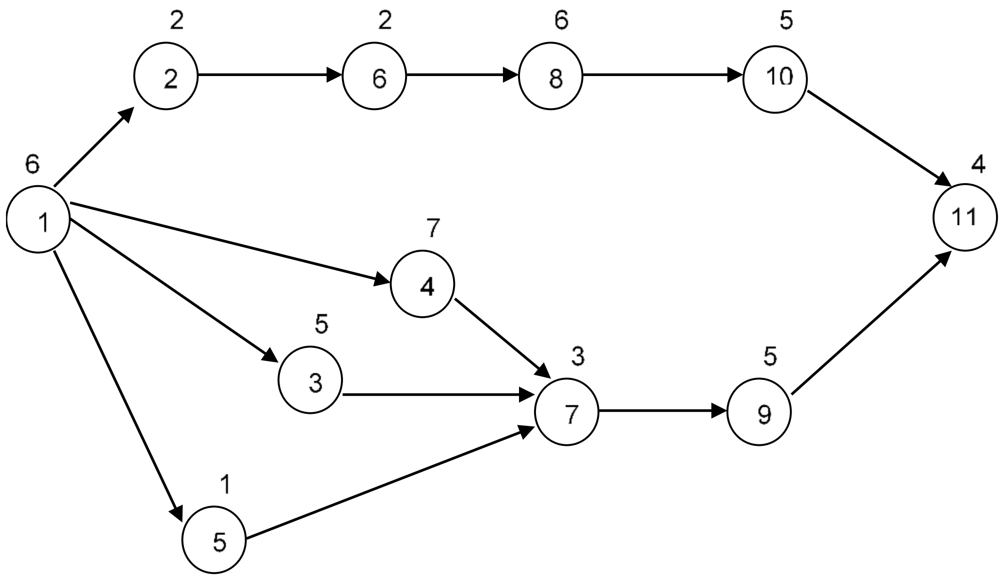


Figure 1.3: Typical Precedence Network Diagram

The execution of each task requires certain time, known as task time. This may be deterministic or probabilistic. The cycle time is the time between consecutive releases of the completed assemblies at the end of the line. It is also the total time (maximum time) allocated to each workstation in the assembly line. All workstations have the same cycle time.

The formula for the cycle time is as follows [2]:

$$Cycle\ Time = \frac{Total\ time\ available\ per\ shift}{Production\ volume\ quantity\ per\ shift} \qquad [\ 2\ ]$$

The cycle time and the number of workstations are expected to be inversely proportional. The greater the cycle time, the smaller the number of workstations required; having more work stations results in smaller cycle times. If the objective is to minimize the number of workstations for a given production rate, it is usually referred as a Type I problem. If the goal is to maximize the production rate by minimizing the sum of a given number of workstations, it is referred as a Type II problem.

The formula to compute the balancing efficiency percentage is given below [3]:

$$Efficency = \frac{Sum\ of\ task\ times}{Number\ of\ workstations \times Cycle\ time} \times 100 \qquad [\ 3\ ]$$

In the balancing efficiency formula, the sum of all task times and cycle time is given as an input. The cycle is computed based on a desired production volume of a product to be assembled in the line. The balancing efficiency is the ratio between the sum of the task times and the total time that is provided to execute all the tasks (number of workstations × cycle time). This research studies only the Type I LBP where the goal is to minimize the number of workstations. The formula clearly shows that the fewer the number of workstations, the greater the balancing efficiency and the fewer is the requirement of resources (operators). If the sum of the task times and the total time provided to execute them were the same, there would be an efficiency of 100%. The prime objectives of the assembly LBP are the following:

- To subdivide the tasks in a given precedence network into a number of workstations for a given cycle time subject to the following two constraints such that the balancing efficiency is maximized (Type I problem):
    - non-violation of the precedence constraints among the tasks, and
    - processing time sums of the tasks assigned to each workstation are less than or equal to the given cycle time.
- To subdivide the tasks in a given precedence network into a given number of workstations without violating the precedence constraints among the tasks such that the cycle time is minimized (Type II problem).

CHAPTER 2

LITERATURE REVIEW

2.1 Line-Balancing Problem Classification

Line-balancing problems can be classified by three parameters that differentiate them. They can be classified by the nature of task times, number of models produced, and the flow shape. The number of models refers to the different versions of a product. Each of these parameters creates two categories, making a total of eight different types of LBP. The nature of task times can be either deterministic or probabilistic; they differ in the variability of the task times. The probabilistic variation of task times is characterized by a probability distribution, while the deterministic task times can be approximated by a fixed value. The number of models produced is separated as single model (one model) and multi model (more than one model). The flow shape can be either straight (line) or U-shaped.

This results in the following types:

1. Single-model deterministic straight-type problem
2. Single-model deterministic U-shape problem
3. Single-model probabilistic straight-type problem
4. Single-model probabilistic U-shape problem
5. Multi-model deterministic straight-type problem
6. Multi-model deterministic U-shape problem
7. Multi-model probabilistic straight-type problem
8. Multi-model probabilistic U-shape problem

The LBP classification scheme is shown in Figure 2.1:

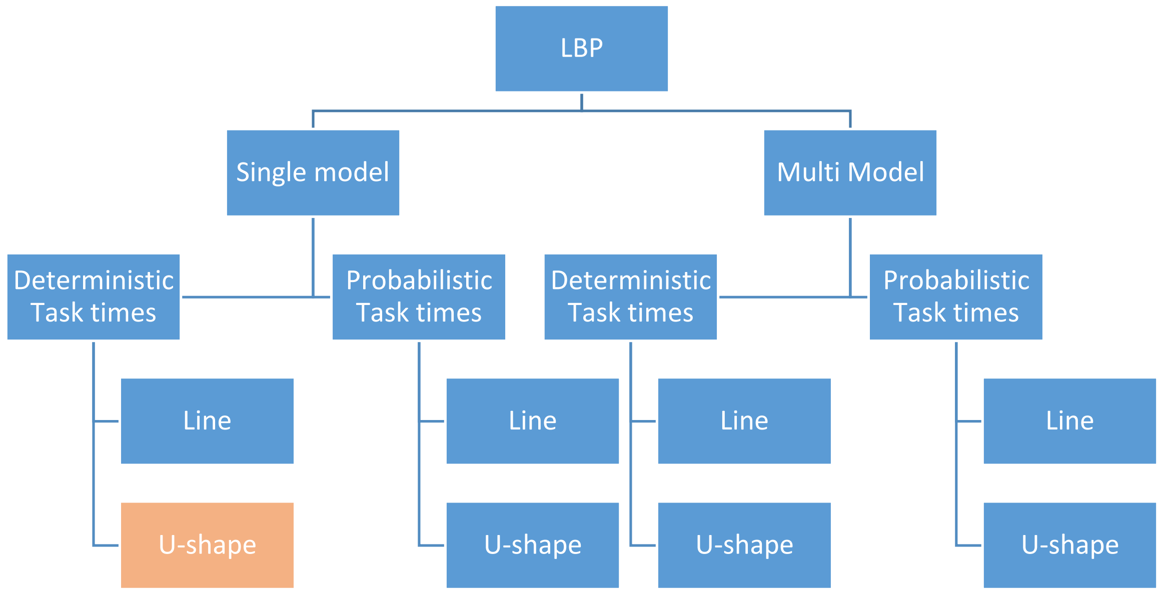


Figure 2.1 Line Balancing Problem Classification

This research focuses on the single model, deterministic, U-shaped LBP.

2.2 Single-Model Deterministic U-shape LBP

Miltenburg and Wijngaard (1994) produced the first publication about the U-shaped LBP. They pointed out an approach to balancing systems arranged in a U-shape way. They showed that this kind of arrangement of the LBP produces a more complex problem than the one dealt with in the traditional LBP. This is because tasks can be assigned to workstations by moving forward, backward, or simultaneously in the precedence network. Following this publication, researchers have been trying to solve this NP-hard problem by using different approaches.

The literature for this type of problem is classified based on the methods that have been previously used to attempt to solve it (Sivasankaran and Shahabudeen, 2014). These methods are as follows:

- Exact Solutions
  - Enumeration procedure

    - Integer programming
    - Shortest path algorithm
    - Other exact solution approaches

- Heuristics
    - Simulated annealing algorithm
    - Ant colony optimization algorithm
    - Multi-pass random assignment algorithm
    - Critical path method
    - Other heuristic approaches
- Stochastic Methods

2.2.1 Exact Solutions

2.2.1.1 Enumeration Procedure

Suppose the following precedence network has a cycle time of six, as represented in Figure 2.2.

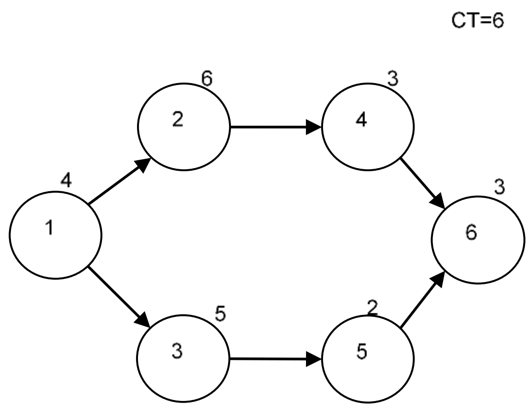


Figure 2.2: Enumeration Process Precedence Network

To solve the U-shape LBP, it is necessary to find the solution that produces the fewest workstations. Recall that a U-shape LBP can move from left to right as well as from right to left. Thus, this problem can begin with either operation 1 or operation 6. Table 2.1 bellow shows these two options.

Table 2.1: Enumeration Process Workstation 1

	V	Assign	Tr	V_2
Ws1	1, 6	1	2	0
	1, 6	6	3	4
				5

This table represents the first workstation. *V* indicates the available operations to assign, *Tr* is the time remaining after assigning an operation, and V_2 is the next operation/s that can be completed with the remaining time. Tasks can continue to be assigned until the remaining time is no longer sufficient for the next task. If the remaining time is less than the task time of the next operations, they cannot be completed and must move on to the next workstation. As seen in the table, the two options lead to two different outcomes that branch the problem in two directions. This is only the start: every workstation may branch into two or more options.

Continuing with the first option to the second workstation, named Workstation 2 of Option 1, the available tasks are now 2, 3 or 6, as seen in Table 2.2.

Table 2.2: Enumeration Process Workstation 2

	V	Assign	Tr	V_2	Tr2
Ws 2 Op 1	2, 3, 6	2	0	0	0
	2, 3, 6	3	1	0	0
	2, 3, 6	6	3	4	0
				5	1

Workstation 2 branches off into four more options. The third workstation for the first option shown in the above table would result in the branches shown in Table 2.3. At this point, tasks 1 and 2 have been assigned. Continuing this procedure with the first option (operation 3), the next step would be as shown in Table 2.4.

Table 2.3: Enumeration Process Workstation 3

	V	Assign	Tr	V_2	Tr2
Ws 3 Op 1.1	3, 4, 6	3	1	0	0
	3, 4, 6	4	3	6	0
	3, 4, 6	6	3	4	0
				5	1

Table 2.4: Enumeration Process Workstation 4

	V	Assign	Tr	V_2	Tr2
Ws 4 Op 1.1.1	4, 5, 6	4	3	5	1
				6	0
	4, 5, 6	5	4	4	1
				6	1
	4, 5, 6	6	3	4	0
				5	1

At this point, there will only be one operation remaining for every option. Following option one again, the final work station would be as shown in Table 2.5.

Table 2.5: Enumeration Process Workstation 5

Ws 5 Op 1.1.1.1	V	Assign	Time remaining	V_2
	6	6	3	0
Ws 5 Op 1.1.1.2	V	Assign	Time remaining	V_2
	5	5	4	0
Ws 5 Op 1.1.1.3	V	Assign	Time remaining	V_2
	6	6	3	0
Ws 5 Op 1.1.1.4	V	Assign	Time remaining	V_2
	4	4	3	0
Ws 5 Op 1.1.1.5	V	Assign	Time remaining	V_2
	5	5	4	0
Ws 5 Op 1.1.1.6	V	Assign	Time remaining	V_2
	4	4	3	0

This process would have to be continued for all possible solutions. In Appendix B, the complete process is shown. The path followed out of all possible paths is shown in Figure 2.3.

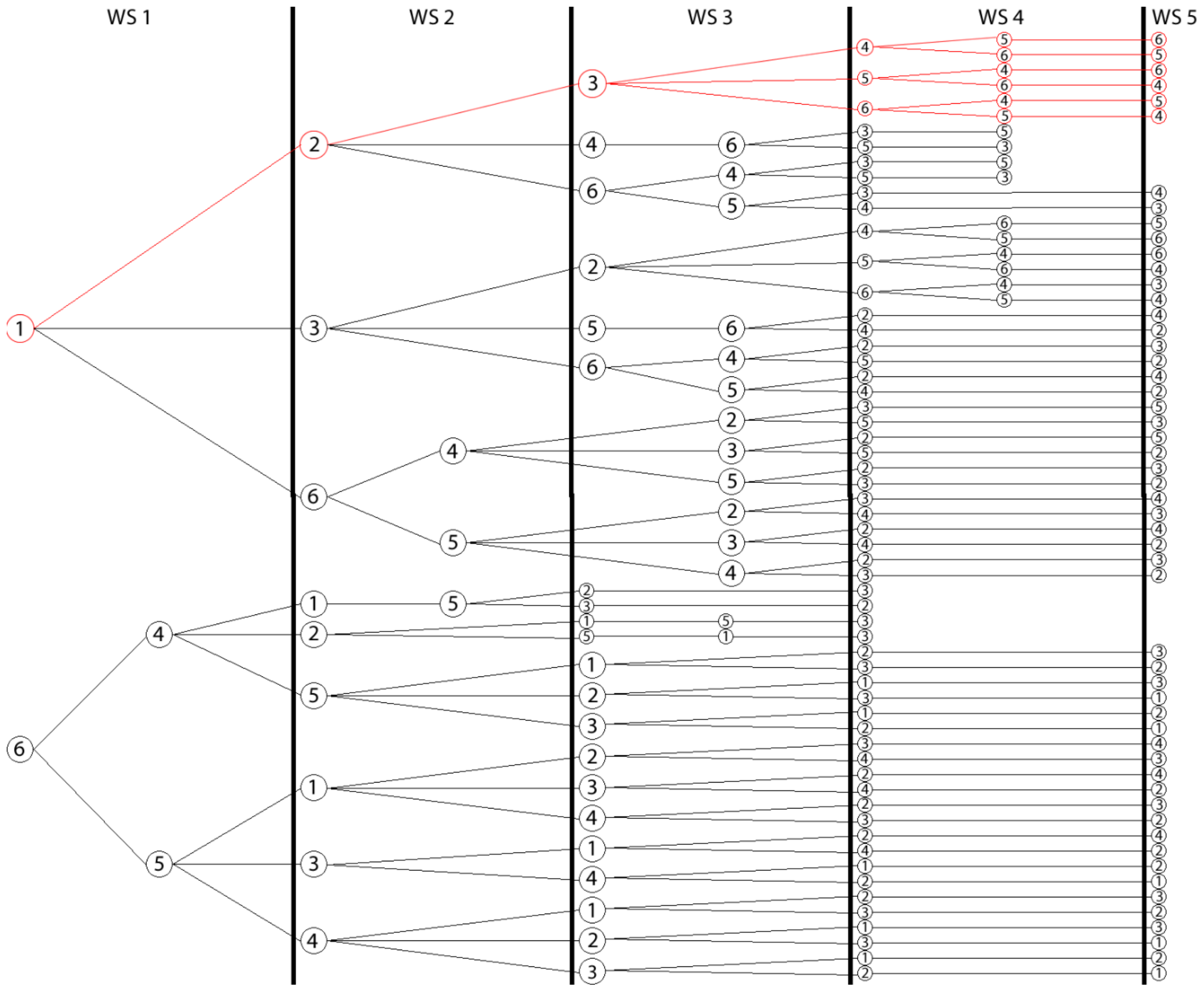


Figure 2.3 Enumeration Process Diagram

There are a total of eight optimal solutions to this problem that require only four workstations. A large number of possible solutions have been obtained from a simple problem. In a more complicated problem, it would be significantly more difficult and less convenient to manually look for every possible outcome and select the best one. A branch and bound approach can help to improve this process. In the branch and bound procedure, workstations are evaluated to determine which branch has better feasibility than the others. This way it is not necessary to evaluate every single branch. An example of this is demonstrated by Miralles (2008).

2.2.1.2 Integer Programming

Integer programming is a field that uses a mathematical optimization, or feasibility program, in which some or all of the variables are restricted to be integers. Solving the LBP using this approach can result in a very large and long problem, even with a small precedence diagram. To understand how this method is used to solve the LBP, an example is presented below.

The assembly of a certain product requires the balancing of a line. Cycle time and task times are given in Table 2.6.

Table 2.6: Task Times for Integer Programming Problem

Task	Time
1	40
2	75
3	50
4	35
5	80
CT	100

This is a Type ILBP with the objective to minimize the number of workstations. The precedence network for this problem is shown in Figure 2.4.

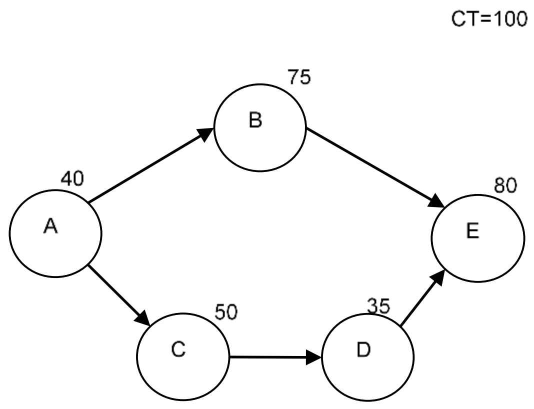


Figure 2.4: Precedence Network for Integer Programming Problem

Let X_{ij} = 1 if task *i* goes to station *j*, otherwise $X_{ij} = 0$.

P_{ij} is a weighted value that is used to ensure the least number of workstations will be used by giving priority in the objective function to the smallest workstation number.

C is the cycle time.

The general mathematical formulation for a U-shaped LBP Type I is shown below [4]:

$$Minimize\ Z = \sum_{i=1}^{n}\sum_{j=1}^{m} P_{ij}X_{ij} \qquad [\ 4\ ]$$

subject to:

Cycle time

$$\sum_{i=1}^{n} T_i X_{ij} \le C\ \forall\ j \qquad [\ 5\ ]$$

Unit Assignment

$$\sum_{j=1}^{m} X_{ij}\ \forall\ i \qquad [\ 6\ ]$$

Precedence

$$X_{kj} \le \sum_{j=1}^{m} X_{ij}\ \forall\ i, k \qquad [\ 7\ ]$$

Expanding the objective function where *i = a, b, c, d, e and j = 1, 2, 3, 4, 5* assigning positional weights one, two, three, four, and five to each workstation resulting in the following:

$$Minimize\ Z = X_{a1} + 2X_{a2} + 3X_{a3} + 4X_{a4} + 5X_{a5} + X_{b1} + 2X_{b2} + 3X_{b3} + 4X_{b4} + 5X_{b5} + X_{c1} + 2X_{c2} + 3X_{c3} + 4X_{c4} + 5X_{c5} + X_{d1} + 2X_{d2} + 3X_{d3} + 4X_{d4} + 5X_{d5} + X_{e1} + 2X_{e2} + 3X_{e3} + 4X_{e4} + 5X_{e5}$$

Applying the cycle time restriction results in:

$$40X_{a1} + 75X_{b1} + 50X_{c1} + 35X_{d1} + 80X_{e1} \leq 100$$

$$40X_{a2} + 75X_{b2} + 50X_{c2} + 35X_{d2} + 80X_{e2} \leq 100$$

$$40X_{a3} + 75X_{b3} + 50X_{c3} + 35X_{d3} + 80X_{e3} \leq 100$$

$$40X_{a4} + 75X_{b4} + 50X_{c4} + 35X_{d4} + 80X_{e4} \leq 100$$

$$40X_{a5} + 75X_{b5} + 50X_{c5} + 35X_{d5} + 80X_{e5} \leq 100$$

After applying unit assignment restrictions, the following is revealed:

$$X_{a1} + X_{a2} + X_{a3} + X_{a4} + X_{a5} = 1$$

$$X_{b1} + X_{b2} + X_{b3} + X_{b4} + X_{b5} = 1$$

$$X_{c1} + X_{c2} + X_{c3} + X_{c4} + X_{c5} = 1$$

$$X_{d1} + X_{d2} + X_{d3} + X_{d4} + X_{d5} = 1$$

$$X_{e1} + X_{e2} + X_{e3} + X_{e4} + X_{e5} = 1$$

Finally, the precedence restrictions are:

$$X_{b1} \leq X_{a1}$$

$$X_{b2} \leq X_{a1} + X_{a2}$$

$$X_{b3} \leq X_{a1} + X_{a2} + X_{a3}$$

$$X_{b4} \leq X_{a1} + X_{a2} + X_{a3} + X_{a4}$$

$$X_{b5} \leq X_{a1} + X_{a2} + X_{a3} + X_{a4} + X_{a5}$$

Since *a* is the immediate predecessor of *c*, then the following occurs:

$$X_{c1} \leq X_{a1}$$

$$X_{c2} \leq X_{a1} + X_{a2}$$

$$X_{c3} \leq X_{a1} + X_{a2} + X_{a3}$$

$$X_{c4} \leq X_{a1} + X_{a2} + X_{a3} + X_{a4}$$

$$X_{c5} \leq X_{a1} + X_{a2} + X_{a3} + X_{a4} + X_{a5}$$

Likewise, *c* is the predecessor of *d*, so:

$$X_{d1} \leq X_{c1}$$

$$X_{d2} \leq X_{c1} + X_{c2}$$

$$X_{d3} \leq X_{c1} + X_{c2} + X_{c3}$$

$$X_{d4} \leq X_{c1} + X_{c2} + X_{c3} + X_{c4}$$

$$X_{d5} \leq X_{c1} + X_{c2} + X_{c3} + X_{c4} + X_{c5}$$

The final restriction says that *e* has two predecessors, *b* and *d*. Applying these restrictions results in the following:

$$X_{e1} \leq X_{b1} + X_{d1}$$

$$X_{e2} \leq X_{b1} + X_{d1} + X_{b2} + X_{d2}$$

$$X_{e3} \leq X_{b1} + X_{d1} + X_{b2} + X_{d2} + X_{b3} + X_{d3}$$

$$X_{e4} \leq X_{b1} + X_{d1} + X_{b2} + X_{d2} + X_{b3} + X_{d3} + X_{b4} + X_{d4}$$

$$X_{e5} \leq X_{b1} + X_{d1} + X_{b2} + X_{d2} + X_{b3} + X_{d3} + X_{b4} + X_{d4} + X_{b5} + X_{d5}$$

A branch and bound procedure is used to solve this system of equations and inequalities, and it is determined that the number of workstations required is four. The tasks are assigned to the workstations as shown in Table 2.7.

Table 2.7: Integer-Programming Problem Results

Workstation	Task(s)
1	A, C
2	B
3	D
4	E

This rather small problem reveals that30 constraints and 25 variables were generated. As a problem becomes more complex and longer, the process for solving it becomes very time consuming and computationally expensive. For any problem, the maximum number of constraints and variables can be calculated with the following formulas [8]:

$$all\ constraints = i * pr + 2i \quad [8]$$
$$all\ variables = i^2$$

where *i* is the number of tasks and *pr* is the number of precedence requirements.

An integer programming solution to this example problem is listed in Appendix C.

2.2.1.3 Dynamic Programming and Branch and Bound

Exact solution procedures for the straight LBP are based on either dynamic programming or branch and bound procedures. The first dynamic programming procedure developed by Jackson (1956) and modified by Held .et al. (1963), subdivides the solution process in stages that correspond to stations. Following Jackson and Held’s contributions, there have been other researchers who have developed dynamic programming approaches, like Gutjahr and Nemhauser (1964), Schrage and Baker (1978), and Easton et al. (1989). The branch and bound procedures are divided into those that are station oriented or task oriented. Some examples of these branch and bound procedures for the straight LBP are Hoffman (1993), Berger et al. (1992), Johnson (1993), and Sprecher (2003). After these publications, many of the methods have been adapted for the U-shaped LBP such as Chun-Hung et al. (2010) for dynamic programming, and Miralles (2008) for branch and bound.

Ogan and Azizoglu (2015) considered a U-shape assembly LBP where each task uses a specific set of equipment and each type of equipment has a specific cost. Their goal was to assign the tasks along with the equipment to the workstations to minimize the total equipment cost. They formulated a mixed-integer linear programming model that was used only for small problems. Also, they proposed a branch and bound algorithm that uses precedence relations and lower bounds. The approach they proposed can be used to solve small-sized problems.

2.2.1.4 Shortest Path Algorithm

Gokcen et al. (2005) modeled the U-shaped LBP as a shortest route model based on the one developed by Gutjahr and Nemhauser (1964) where they used a shortest route in a finite directed network. This approach can be used as the framework for developing a more efficient heuristic to solve the U-shape problem. This approach is not suitable for large problems and it does not show alternative solutions that could also be obtained with the same number of workstations.

2.2.1.5 Other Exact Solution Approaches

Other exact solution methods have been developed, like the goal-programming model to solve the U-shape LBP that was developed by Gocken and Agpak (2006). They used their model to optimize different objectives by considering assignment constraints, cycle-time constraints, workstation constraints, and task-load constraints. They used three goals in their model. The first goal with priority level one is to restrict the maximum number of workstations. The second goal with priority level two, restricts the cycle time to not exceed a given value. The final goal of priority level three stipulates that the number of tasks per workstation should not exceed a fixed value. To test the application of their model, they used numerical examples. In this goal-programming approach, the objective function is constructed using the deviation variables assigned to goal constraints with stated weights. They tested their model using various randomized examples and compared them to the solution of the straight LBP. It was shown that the U-shape model gives better results.

The number of possible solutions in the U-shaped LBP increases greatly as the number of tasks increase. With such a vast search space it is nearly impossible to obtain an efficient solution using a deterministic algorithm, which computes a function on a one-to-one basis, meaning it includes all countable possibilities. However, many attempts have been made to use an exact approach (Held and Karp (1963), Jackson (1956), and Mertens (1967)). None of these methods have proven to be of practical use for large problems due to their computational and time inefficiency.

2.2.2 Heuristics

2.2.2.1 Simulated Annealing

For simulated annealing algorithms, Ozcan and Toklu (2009) developed a hybrid improvement heuristic to the straight and U-shape LBP. This was based on the idea of adaptive learning and simulated annealing to maximize the balancing efficiency and reduce the variation of workloads. The adaptive learning approach could be also be used in other meta-heuristics.

Jayaswal and Agarwal (2014) developed a simulated annealing approach to solve the U-shape LBP by not only assigning workers to workstations, but also by proposing a U-shape LBP that takes into account equipment, arguing that most problems in the literature do not take into account the equipment needed to perform the tasks. They pointed out that it is often desirable to reduce certain task times by assigning more workers or alternative equipment at a given workstation. The problem in such cases is not only to reduce the times, but to assign alternatives (number of workers and equipment type) to the workstations. Research on such resource-dependent assembly LBPs is scarce. The authors address the problem of resource-dependent U-shaped LBPs and proposed a simulated annealing-based metaheuristic. The simulated annealing generates good solutions.

2.2.2.2 Ant Colony Optimization

Ant colony optimization (ACO) algorithms, are inspired by observation of real ant colonies in nature. An interesting behavior of the ants is how they follow the paths between their nests and food sources. As they are traveling through these paths, they leave behind a substance called a pheromone. Ants can detect these pheromones and follow paths that are marked with strong concentrations of pheromones. By following these paths, the pheromone concentration is reinforced and more ants follow that route, this is how a colony of ants finds the shortest route from their nests to the food source.

Zhang et al.(2008) proposed a new design of ant colony optimization to solve the U-shaped LBP so that the number of workstations is minimized. The proposed algorithm uses the trial information that is deposited between the task and its position, and also an adapted version of pheromone summation rules. The proposed method adopts local and global pheromone updating to explore the solutions. Pheromones are used so that a trail can be followed pointing to the different solutions. A pheromone trail refers to the desirability of assigning a task to a workstation. Pheromones are indicators that are left when a path is explored, a path is more likely to be explored if it has a higher concentration of pheromones. Couglu et al. (2009) compared this algorithm with a simulated annealing algorithm and found the ant colony optimization algorithm outperforms the simulated annealing algorithm. To reach an ultimate value solution using this algorithm, its parameter may be optimized using Taguchi's technique.

2.2.2.3 Multi-pass Random Assignment Algorithm

Yegul et al. (2010) saw the U-shaped LBP and considered a new algorithm to minimize the total number of stations for a given cycle time. They found a special two-sided design in which one side was arranged as a U-shape and the other as a traditional straight line. By proposing a multi-pass random assignment algorithm, they could potentially find the minimum number of work stations to assemble a product. Once the initial solution is generated, tasks are classified in five types depending on the direction in which they can be assigned. The multi-pass random assignment consists in changing tasks from one side to the other by randomly selecting one of the five types. There is no statistical evidence presented in this publication to ensure that this heuristic approach performs well and this is the only recent research publication in this area. However, with the few literature sources available, researchers claim that it does in fact perform well.

2.2.2.5 Critical Path Method

Avikal (2013) developed this heuristic-based method to reduce the number of workstations. The problem is solved by treating the network as a project network and using a critical-path method to divide the critical and non-critical activities. Temporary workstations are created to assign the critical activities by priority. These new workstations will be kept if the slack time is minimal. This process continues until all tasks are assigned to a workstation. Avikal (2013) concentrated on the advantages of the U-shape layout in contrast with the straight line model. This is a graph-based method that could be incorporated with other methods to improve their performance.

2.2.2.6 Other Approaches

Several heuristic-based methods for the traditional assembly LBP have been developed. Baybars (1984) and Talbot et al. (1981) review and evaluate these different approaches. Another proposed solution considers a five-phase method using task elimination, decomposition and heuristics (Baybars 1986). Also, a genetic algorithm was used to obtain near optimal solutions to the traditional assembly LBP in combination with heuristic-based methods like the method proposed by Leu, Matheson and Rees (1994). Leu, Matheson and Rees used five heuristic rules to generate the initial population of their proposed genetic algorithm. Then they used two objectives: one was to minimize the mean square idle time and the other was to minimize the mean idle time.

Manavizadeh et al. (2015) proposed a multi-objective approach to solve a U-shaped mixed-model assembly line to minimize the cycle times, the waste in each station and the work overload. To minimize the three objective functions, a heuristic algorithm was designed. They begin the algorithm with an initial solution using branch and bound for each one of the objectives individually. Then, each solution was put into the other objective models and the objective values were found. The combination of these objectives for this kind of problem had not been studied in previous publications. They compared straight line with U-shape line problems so that decision makers can compare the results of both cases and choose the shape of the assembly line they want to use.

2.2.3 Stochastic Methods

In the majority of the cases deterministic problems can be adapted to stochastic problems. Stochastic solutions are rarely used in real problems, they may be implemented when a deterministic model does not give desired results. Stochastic approaches to solve the LBP have been explored using: heuristics (Liu et al., 2005), simulated annealing (Ozcan, 2010, Cakir et al., 2011), genetic algorithm (Baykasoglu and Ozbakir, 2007), and shortest path algorithm (Boysen and Fliedner, 2008). The stochastic solutions are for single model, straight line and U-shape line balancing problems. The execution times of the tasks are probabilistic and in the majority of the applications, a normal distribution is used. The stochastic approach is widely studied but in reality, most industries use standard task times for balancing their assembly lines, this standard time is recalculated periodically so that the assembly line is adjusted and maintained balanced.

A stochastic problem can be solved using a deterministic approach, the main difference is that in a deterministic problem, tasks are assigned to a workstation as long as their task times do not exceed the cycle time. In the stochastic problem, a probability that the station time is not exceeded is calculated. For a stochastic problem where the task times are normally distributed, this probability is calculated by dividing the remaining time after assigning a task to a workstation by the square root of the sum of the variances of tasks assigned (Z-value). Tasks are assigned to a workstation as long as the probability calculated does not exceed the desired probability, otherwise a new workstation must be opened.

CHAPTER 3

PROBLEM STATEMENT

The main objective of the Type 1 U-shaped LBP is to assign tasks to workstations so that the number of stations is minimized. This involves the allocation of tasks subject to capacity and precedence constraints. This is depicted in Figure 3.1 below:

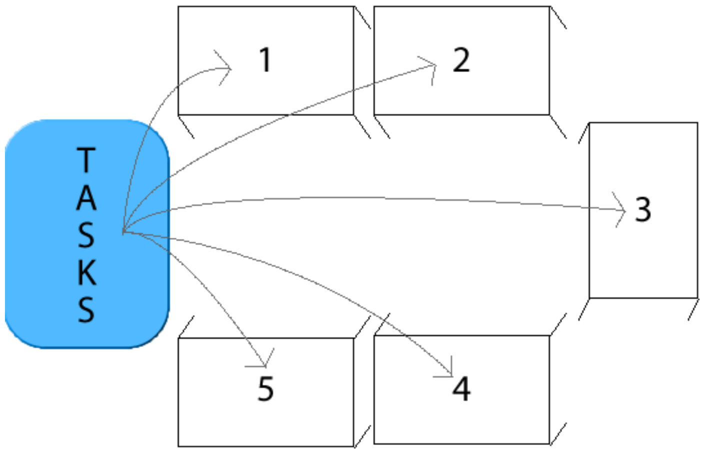


Figure 3.1 Task Assignation to Workstations

The scope of this research is restricted to the Type I U-shaped LBP.

3.1 U- Line Balancing Problem Constraints

The constraints associated with the U-shaped LBP are the following:

- Capacity
- Precedence

Capacity constraints restrict the total number of tasks that can be assigned to a workstation based on the U-line cycle time. The cycle time is established by the user based on the production requirements.

The sum of all task processing times grouped in a workstation is called station load. This load cannot exceed the cycle time to satisfy the demanded requirements. Station load can be calculated using the equation [9]:

$$S_{load} = \sum_{i=1}^{n} t_i \qquad [\ 9\ ]$$

Where *n* is the number of tasks assigned to a workstation. The theoretical minimum number of workstations (without taking into account precedence restrictions) can be calculated as follows [10].

$$WsMin = \frac{\sum_{i=1}^{N} t_i}{CT} \qquad [\ 10\ ]$$

Where *t* is the task time and *N* is the total number of tasks in a given problem. The time remaining in a work station is $Tr = CT\text{-}S_{load}$.

3.2 Precedence Constraints

Due to the technical characteristics of products, manufacturing operations must follow a certain order. These are called precedence constraints. All these relationships create what is called a precedence network. This network can be represented as a diagram. An example of a precedence diagram is illustrated in Figure 3.2.

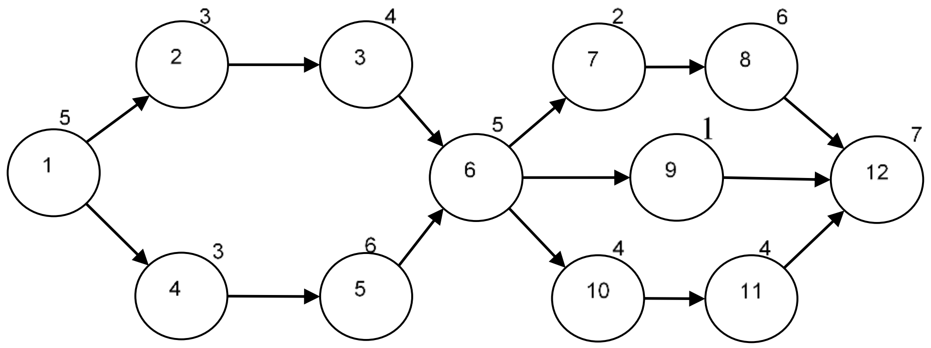


Figure 3.2 Precedence Diagram

The assignment of tasks to workstations must satisfy these precedence relationships. For the traditional LBP the precedence relationship must be followed in a single direction, but, as stated by Miltenburg and Wijngaard (1994), the U-shaped LBP, allows for precedence relationships to be satisfied from left to right and also from right to left. This produces a more complex problem than when production systems are arranged in a line.

3.4 Two Genetic Algorithms

In complexity theory, problems can be categorized by their difficulty and the time required to solve them. There are four general categories. These categories, listed in increasing difficulty, are P, NP, NP-Complete and NP-hard, where P is the easiest (solvable in P time) and NP-hard is the most difficult one. The traditional (linear) assembly LBP is known to be NP-hard. According to Debora and Wainwright (1986), if there are *m* tasks and r ordering constraints then there are $m!/2^r$ possible tasks sequences.

The U-shaped LBP has been an attractive topic among researchers since 1994. Nonetheless, there is still a vast area of undiscovered knowledge. This research introduces two genetic algorithms, one direct-coded and the other indirect-coded that contribute to the development of solutions to the NP-hard problem. These genetic algorithms obtain good solutions and sometimes optimal solutions addressing two objectives—the first objective is to minimize the number of workstations required and the second objective is to minimize a smoothness index. The efficiency of the algorithms is increased by separating the problems according to their size (small, medium, large). This creates a specific model to a range of problems that results in reducing the computational effort required to arrive at solutions. These two genetic algorithms are further explained in chapters 5 and 6.

CHAPTER 4

SOLUTION TO THE U-SHAPE BALANCING PROBLEM USING HEURISTIC RULES

Many heuristic approaches can be found in the literature to solve the simple LBP. Some of the most popular techniques are discussed by Talbot, Patterson and Gehrlein (1980). All of this research is based on the traditional LBP (straight line). The heuristic rules from the traditional in-line balancing problem were adapted by Martinez and Duff (2004) so that they could now be used in a U-shape LBP.

4.1 Heuristic Rules to Solve the U-Shaped LBP

Ten heuristic rules were adopted for their use in this research to find solutions to the Type I U-shaped LBP. All these heuristic rules were previously used to solve the simple LBP. However, some modifications were made to allow them to work for the U-shaped LBP. In the original heuristic rules, a task could be assigned only if all of its predecessors had been assigned. However, in the modified heuristic rules, a task can be assigned if all the predecessors or the successors of a given task have been assigned. This allows for tasks to be assigned following the precedence diagram from left to right but also from right to left.

The heuristic rules proposed in this research generate weights for every task, these weights are used during the task assignment process. The value of the weight for the tasks determine which tasks have more priority to be assigned to a workstation from the set of assignable tasks.

Following are ten heuristic rules used in this research. The first rule was posted by Miltenburg and Wijngaard (1994). The other nine heuristic rules are introduced in this research for solving the U-shaped LBP.

1. The Modified Ranked Positional Weight, Miltenburg and Wijngaard (1994), 2. Maximum Total Number of Follower Tasks or Precedence Tasks,

3. Minimum Total Number of Follower Tasks or Precedence Tasks,

4. Maximum Task Time,

5. Minimum Task Time,

6. Maximum Number of Immediate Followers or Immediate Precedence Tasks,

7. Minimum Number of Immediate Followers or Immediate precedence Tasks,

8. Minimum U-line Upper Bound,

9. Minimum U-line Lower Bound, and

10. U-line Minimum Slack.

These heuristic rules are described in detail below:

Let μ_k^p be the set of tasks that must precede task *k*, and μ_k^s be the set of tasks which must succeed task *k*. Then, at any time the set of assignable tasks, $V = \{k \mid all\ i \in \mu_k^p\ or\ all\ j \in \mu_k^s\}$ have already been assigned.

1. Maximum Ranked Positional Weight

The priority function *p(k),* called the U-line Maximum Ranked Positional Weight, is defined as [11]:

$$p(k) = max\left\{t(k) + \sum_{i \in \mu_k^p} t(i), t(k) + \sum_{i \in \mu_k^s} t(j)\right\} \qquad [\ 11\ ]$$

2. Maximum Total Number of Follower Tasks or Precedence Tasks

The priority function $p_{maxnfk}(k)$, called the U-line Maximum Total Number of Follower or Precedence Tasks [12]

$$p_{\text{maxnfk}}(k) = \max\{number-of-tasks \in \mu_k^p, number-of-tasks \in \mu_k^s\} \qquad [\ 12\ ]$$

3. Minimum Total Number of Follower Tasks or Precedence Tasks

The priority function $p_{minnfk}(k)$, called the U-line Minimum Total Number of Follower or Precedence Tasks [13]

$$p_{\text{minnfk}}(k) = \min\{number-of-tasks \in \mu_k^p, number-of-tasks \in \mu_k^s\} \qquad [\ 13\ ]$$

4. Maximum Task Time

The priority function $p_{Mtk}(k)$, called the U-line Maximum Task Time [14]

$$P_{\text{Mtk}}(k) = t(k) \qquad [\ 14\ ]$$

5. Minimum Task Time

The priority function $p_{mtk}(k)$, called the U-line Maximum Task Time [15]

$$P_{\text{mtk}}(k) = t(k) \qquad [\ 15\ ]$$

6. Maximum Number of Immediate Followers or Immediate Precedence Tasks

Let μ_k^{ip} be the set of tasks that must immediately precede task *k*, and μ_k^{is} be the set of tasks that must immediately succeed task *k*.

The priority function $p_{maxipis}(k)$, called the U-line Maximum Number of Immediate Followers or Immediate precedence Tasks. [16]

$$p_{\text{maxipis}}(k) = \max(\mu_k^{ip}, \mu_k^{is} \} \qquad [\ 16\ ]$$

7. Minimum Number of Immediate Followers or Immediate Precedence Tasks

The priority function $p_{minipis}(k)$, called the U-line Maximum Number of Immediate Followers or Immediate Precedence Tasks. [17]

$$p_{\text{minipis}}(k) = \max(\mu_k^{ip}, \mu_k^{is} \} \qquad [\ 17\ ]$$

8. Minimum U-line Upper Bound

Let $(x)^+$ be the least integer $>x$ and c be the cycle time.

The priority function $p_{ub}(k)$, called the U-line Upper Bound [18]

$$p_{ub}(k) = \min\left[N+1-\left((t(k)+\sum_{i\in\mu_k^s} t(i))/c\right)^+, N+1-\left((t(k)+\sum_{j\in\mu_k^p} t(j))/c\right)^+ \right] \qquad [\ 18\ ]$$

9. Minimum U-line Lower Bound

The priority function $p_{lb}(k)$, called the U-line Lower Bound [19]

$$P_{lb}(k) = \min\left[\left((t(k)+\sum_{i\in\mu_k^s} t(i))/c\right)^+, \left((t(k)+\sum_{j\in\mu_k^p} t(j))/c\right)^+ \right] \qquad [\ 19\ ]$$

10. U-line Minimum Slack

The priority function $P_{slack}(k)$, called the U-line Minimum slack [20]

$$P_{slack}(k) = p_{ub}(k) - p_{lb}(k) \qquad [\ 20\ ]$$

4.2 Assigning Tasks to Work Stations

The step-by-step procedure is given below:

1. Read the data: task time, task number, cycle time, and precedence relations.
2. Compute the weights for each task using the desired heuristic rule.
3. Rank the task based on the weights computed in Step 2 and give the same rank for the tasks whose weights are equal.
4. Determine the set of assignable tasks V and assign the task with the best rank calculated in Step 3. If a tie occurs, resolve it using the maximum task time. If it does not resolve the tie, use the maximum task number.

5. Has the cycle time at the station been completely filled? Is no one task time from *V* less than or equal to the remaining time in the station? If yes go to Step 6. If no, go to Step 4.

6. Is $V = \emptyset$? If yes, stop. If no, open a new station and go to Step 4.

An illustration of this assignment process uses heuristic rule 2: Maximum Total Number of Follower Tasks or Precedence Tasks and the Jackson's Problem (Table 4.1). The assignable tasks for the first station are $V= \{1,11\}$. Since $p_{maxnfk}(1\ )=10$ and $p_{maxnfk}(11\ )=10$..The tie can be broken by using the maximum task time, $max(t(1),t(11))=max(6,4)=6$, and the assignment of task 1, then $V=\{11,2,4,3,5$. As task 11 has the highest priority and sufficient cycle time remaining, it is also assigned to workstation 1. The remaining assignment process is described in Table 4.1.

Table 4.1: Task Assignment Process for the Jackson's Problem Using Heuristic Rule 2 Maximum Total Number of Follower Tasks or Precedence Tasks

Station	V	Will fit station	Rank	Assigned	Time remaining in station
1	1 1	1 11	1 1	1	4
	1				
	2 3 4 5 11	2 5 11	3 4 1	11	0
2	2 3 4 5 9 10	2 3 4 5 9 10	3 4 4 4 2 3	9	5
	2 3 4 5 7 10	2 3 5 7 10	3 4 4 3 3	10	0
3	2 3 4 5 7 8	2 3 4 5 7 8	3 4 4 4 3 4	7	7
	2 3 4 5 8	2 3 4 5 8	3 4 4 4 4	2	5
	3 4 5 6 8	3 5 6	4 4 4	3	0
4	4 5 6 8	4 5 6 8	4 4 4 4	4	3
	5 6 8	5 6	4 4	6	1
	5 8	5	4	5	0
5	8	8	4	8	4

4.3 Computational Results Using the Heuristic Rules

Eight sets of LBPs representative of various problem types were taken from the literature. Each problem consists of a precedence diagram, tasks times and cycle time. Since each problem is solved with different heuristic rules, this set of cases can be considered to constitute 80 problems.

For each of the eight problems, line balance solutions were obtained using the ten heuristic rules described above (see Tables 4.2 through 4.5).

Table 4.2: Results Obtained for Aase's and Bowman's Problems

Heuristic Rule	Problem	

	Aase
	Tasks=9
	CT=50
		Bowman
		Tasks=8
		CT=20

1	5-30[1]	4-9
2	5-35	4-8
3	5-20	4-9
4	5-15	4-8
5	5-45	4-17
6	5-35	4-8
7	5-20	4-9
8	5-0[2]*	4-8
9	5-20	4-9
10	5-0*	4-5

[1] 5-30 denotes 5 stations plus 30 times units on a sixth station.
[2] 5-0 Denotes that the time remaining in station 5 is zero.
* Denotes that the minimum number of stations (sum of all task times/cycle time) was found.

Table 4.3: Results Obtained for Dar-El's and Jackson's Problems

Heuristic Rule	Problem	

	Dar-El
	Tasks=11
	CT=48
		Jackson
		Tasks=11
		CT=10

1	3-42*	4-9*
2	3-45*	4-8*
3	4-8	4-9*
4	3-44*	4-8*
5	4-45	4-17*
6	3-45*	4-8*
7	4-8	4-9*
8	3-42*	4-8*
9	3-44*	4-9*
10	3-47*	4-5*

Table 4.4: Results Obtained for Johnson's and Ponnambalam Aravindan and Naidu's Problems

Heuristic Rule	Problem	
	Johnson Tasks=5 CT=45	Ponnambalam Aravindan and Naidu Tasks=12 CT=10
1	4-0*	5-5
2	4-0*	5-6
3	4-0*	5-9
4	3-40*	5-5
5	4-0*	6-7
6	4-0*	5-7
7	4-0*	5-5
8	4-0*	5-5
9	4-0*	5-5
10	4-0*	5-5

Table 4.5: Results Obtained for Scholl and Klein's and Tonge's Problems

Heuristic Rule	Problem	
	Scholl and Klein Tasks=12 CT=10	Tonge Tasks=21 CT=20
1	6-6	5-7*
2	6-0*	5-7*
3	6-8	5-12*
4	6-7	5-5*
5	8-7	5-13*
6	6-6	5-11*
7	6-7	5-5*
8	6-0*	5-7*
9	6-8	5-12*
10	6-7	5-10*

The precedence network diagrams for the eight problems can be found in Appendix A. The number of times in which a heuristic rule produced results achieving the minimum number of work stations are summarized on Table 4.6.

Table 4.6: Summary of Heartstring Rules Producing Balances Achieving the Minimum Number of Workstations

Rule	Times achieving minimum number of workstations
1	4
2	5
3	3
4	4
5	3
6	4
7	3
8	6
9	4
10	5

These results show that heuristic rule 8 produced results achieving the minimum number of workstations six times. Heuristic rules 2 and 10 produced results achieving the minimum number of work stations five times. Heuristic rules 1, 4, 6 and 9 produced results achieving the minimum number of workstations four times. Finally, heuristic rules 3, 5 and 7 produced results achieving the minimum number of workstations three times.

CHAPTER 5

INDIRECT-CODED GENETIC ALGORITHM

In a genetic algorithm coded indirectly, the solution does not necessarily represent a solution in reality. The solutions require external information and a non-trivial translating process to be used in real life. Usually, indirect coding is computationally pricier in time and memory. Nonetheless, it can be more effective as it contains more knowledge about the problem.

In this indirect-coded genetic algorithm, each gene in the chromosome represents a heuristic rule and the length of the chromosome may vary. This is because the genes do not represent the tasks themselves but rather, they provide rules as to how they will be assigned. A more detailed explanation is provided in section 5.4.

The heuristic rules described in Chapter 4 can produce good solutions for the U-shaped LBP, however, there is nothing that guarantees that will be the case. Six new heuristic rules are introduced to this procedure. These rules are as listed below:

11. Minimum sum of following task times [21]:

$$p_{minsk}(k) = \min\left\{\sum_{i\in\mu_k^s} t(i)\right\} \qquad [\ 21\ ]$$

12. Maximum sum of following task times [22]:

$$p_{maxsk}(k) = \max\left\{\sum_{i\in\mu_k^s} t(i)\right\} \qquad [\ 22\ ]$$

13. Minimum sum of preceding task times [23]:

$p_{minpk}(k) = \min\left\{\sum_{i \in \mu_k^p} t(i)\right\}$ [23]

14. Maximum sum of preceding task times [24]:

$p_{maxpk}(k) = \max\left\{\sum_{i \in \mu_k^p} t(i)\right\}$ [24]

15. Random minimum weight [25]:

P_{nrand} = min R [25]

where *R* is a random integer between *1* and *N* and *N* is the total number of tasks.

16. Random maximum weight [26]:

P_{nrand} = max R [26]

where *R* is a random integer between *1* and *N* and *N* is the total number of tasks.

One way to get improved solutions using these heuristic rules is to use a number of rules simultaneously to break ties during the task assignment process. It can be done with the implementation of a genetic algorithm. Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. There are three basic operators found in every genetic algorithm: reproduction, crossover and mutation.

5.1 Reproduction

The process of selection is done starting with a population of ordered chromosomes according to their value of aptitude (number of workstations) and smoothness index. In this step of the process, the percentage of chromosomes to which the crossover operation will be applied is selected, as well as the chromosomes that will pass on to the next generation before the mutation process.

5.2 Crossover

The whole idea of creating new generations of chromosomes through genetic operations is to find better solutions than the ones in the initial population. In a nutshell, crossover is to select two parent chromosomes and combine their characteristics to produce new child chromosomes. The number of child chromosomes produced is equal to the number of parents involved in the crossover. The process of crossover used for this research is presented below.

Suppose two parent chromosomes of equal length have been selected, as shown in Figure 5.1. Two crossover points are selected represented by the dashed line.

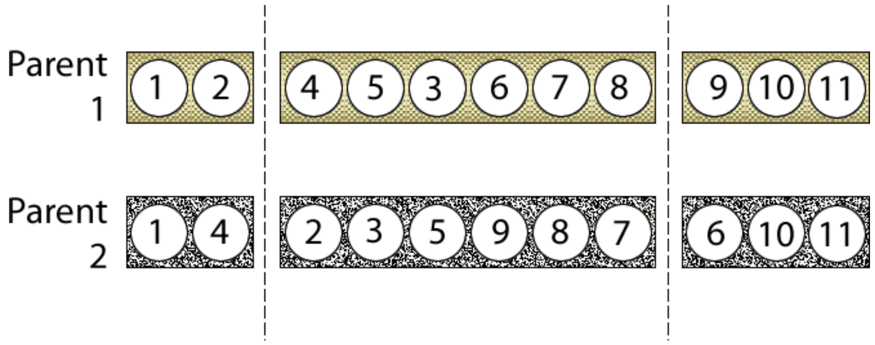


Figure 5.1: Crossover Parents

To create the child chromosomes, the middle parts of the two parents must be swapped. The resulting children are depicted in Figure 5.2.

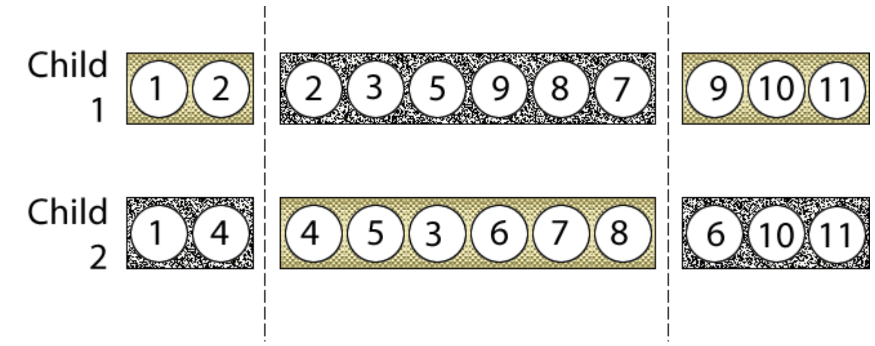


Figure 5.2: Crossover Children

These children are similar to their parents in that they preserve the start and end portions of the chromosome. The purpose of this method is to search for solutions that are similar to those of the parent chromosomes. Searching for around chromosomes that are more fit will most likely result in another optimal solution, but this is not always the case.

5.3 Mutation

Mutation is an operation that will be applied to every chromosome that was previously selected to mutate. Its purpose is to diversify the population so that the solutions don't come from the same traits every time. This allows for a broader search of all possible chromosomes. The gene in which a chromosome is mutated is selected randomly. The place where this gene will be replaced is also selected at random. The rest of the chromosome is preserved as it was before. An example of this is shown in Figure 5.3.

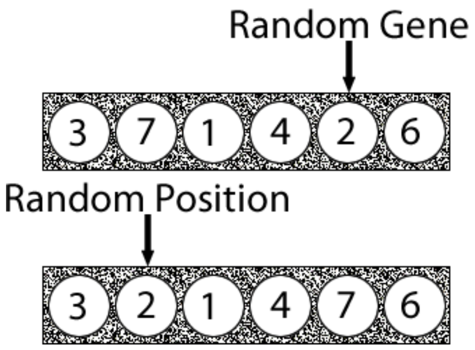


Figure 5.3: Mutation Example

5.4 Decodification

As mentioned earlier, an indirect-coded genetic algorithm consists of chromosomes made up by genes that represent a heuristic rule. For example, in a chromosome that looks like the one on Figure 5.4, each gene doesn't tell which task to assign, but rather, indicates which heuristic rule to use to assign a task.

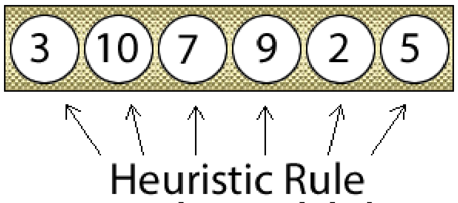


Figure 5.4 Indirect-Coded Chromosome

Tasks are assigned to workstations according to the precedence diagram and using the specified heuristic rule. If a tie occurs between tasks selected, the next gene will be used until ties no longer prevail. However, if a tie cannot be resolved the whole chromosome is unfit and a low fitness value should be assigned to it. This process continues until all tasks have been assigned.

For this genetic algorithm, the following steps are used (based on the method proposed by Ponnambalam, Aravindan and Mogilesswar (2000)) for the simple LBP adapted to the U-shaped LBP).

1. Get the data needed for the LBP: Number of tasks, task number, cycle time, and precedence relation.
2. Initialize the population randomly. Each gene in a chromosome represents one heuristic rule; here the chromosome is using random values between 0 and 16, as 16 heuristic rules are being used.
3. Step-by-step, assign the task to workstations using the heuristic rule number represented by the genes. If a tie occurs, follow with the next gene until the tie is broken. Continue until all genes are exhausted. If during any time of the task assignment process the tie cannot be broken after all genes are exhausted, assign low fitness values to that chromosome so it will not be considered for the next generation. Similarly, assign the task to workstations using the remaining chromosomes.
4. Calculate the values of the two objectives: number of work stations and smoothness index.

5. If an optimal solution or desired result is not obtained in the first interaction, continue with the reproduction crossover and mutation to obtain the new generation and go to Step 5.

5.5 Numerical Example

Jackson's Problem (Figure 5.5) is used to illustrate the procedure.

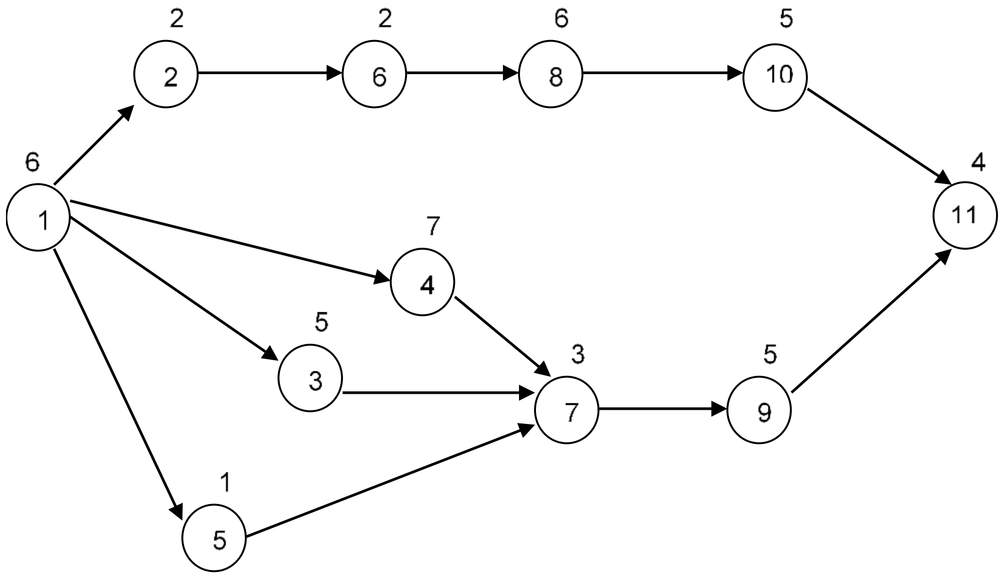


Figure 5.5: Jackson's Problem

All the required information is shown in the figure above: the cycle time for this problem will be of *11 (CT = 11)*, the initial population will be of 20 chromosomes, and the number of genes per chromosome will be 15. Table 5.1 shows the initial population. Using chromosome number one, Table 5.2 shows the solution of the U-shaped LBP.

Table 5.1 Jackson's Problem Initial Population

Chromosomes	
1	11,13,15,14,11,4,16,7,15,16,7,14,11,14,6
2	15,1,5,16,9,12,6,5,8,1,8,14,6,14,15
3	1,4,2,4,1,7,10,1,14,2,3,11,10,13,3
4	6,1,15,8,10,14,12,10,2,11,14,3,10,3,9
5	6,6,16,7,7,8,5,12,13,12,14,10,11,10,16
6	5,16,10,9,4,16,3,6,1,14,16,2,4,5,13
7	13,1,7,5,9,16,2,7,14,14,8,11,14,6,16
8	2,13,2,2,8,4,8,3,6,7,6,14,12,6,1
9	14,6,2,1,14,5,8,1,11,8,16,7,9,16,9
10	1,4,8,10,8,13,9,9,9,12,12,3,1,5,15
11	12,12,1,9,7,14,5,7,13,15,9,7,6,5,5
12	15,13,8,1,4,15,16,12,12,3,6,3,9,6,2
13	9,15,16,9,3,13,4,10,11,6,7,7,3,6,1
14	7,9,13,8,10,10,1,10,2,16,7,12,5,4,8
15	15,3,10,12,4,5,12,4,3,9,14,7,7,8,8
16	4,15,1,1,8,5,4,11,1,5,3,12,15,11,2
17	9,1,11,16,5,12,11,16,7,5,7,12,10,8,2
18	6,12,15,1,2,1,7,7,9,6,4,11,9,4,5
19	5,13,5,16,9,2,15,13,4,9,3,11,10,6,16
20	14,9,8,11,7,8,6,11,8,7,8,6,5,3,10

Table 5.2 Jackson's Problem Solution

Station	V	Will fit	Rank	Rule used	Assigned	Time left
1	1,11	1,11,	1,7,	11	1	5
1	2,3,4,5,11	2,3,5,11,	7,7,7,1,	13	11	1
1	2,3,4,5,9,10	5,	5,	15	5	0
2	2,3,4,9,10	2,3,4,9,10,	2,2,2,7,5,	14		
2	2,3,4,9,10	2,3,4,9,10,	2,4,4,6,6,	11	2	9
2	3,4,6,9,10	3,4,6,9,10,	3,1,6,3,3,	4	4	2
2	3,6,9,10	6,	6,	16	6	0
3	3,8,9,10	3,8,9,10,	2,2,2,2,	7		
3	3,8,9,10	3,8,9,10,	4,3,5,4,	15	8	5
3	3,9,10	3,9,10,	4,4,1,	16	10	0
4	3,9	3,9,	2,2,	7		
4	3,9	3,9,	2,7,	14	3	6
4	7,9	7,9,	5,6,	11	7	3
5	9	9,	7,	14	9	6

V is the set of tasks that satisfy the precedence relation, but only some of the tasks will fit in the workstation with the time remaining. As is shown, the heuristic rules were followed as they appear in the first chromosome. However, tasks were assigned only if there was no tie using that heuristic rule. The final result is depicted in Figure 5.5 below:

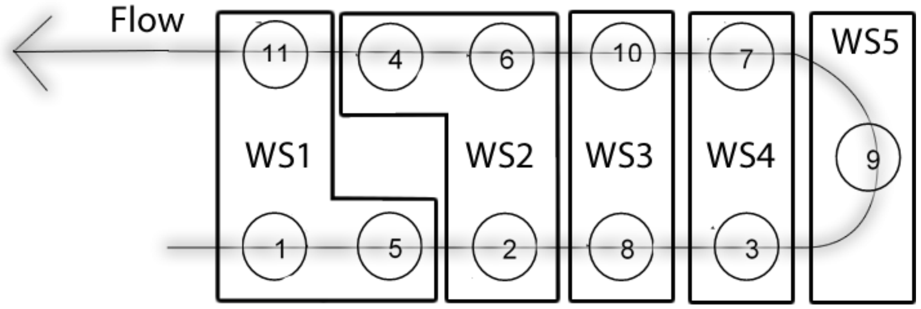


Figure 5.6: Jackson's Problem Solution

CHAPTER 6

DIRECT-CODED GENETIC ALGORITHM

The direct-coded genetic algorithm is fed with entry data specific to each problem. Each problem has its own number of tasks, task times, precedence constraints and cycle times. The direct-coded genetic algorithm differs from the indirect-coded algorithm in that the direct algorithm doesn’t have the number of genes as a variable—the number of genes it uses is the number of tasks that the specific problem has. Once this data is applied, an initial population is generated. Now the search for an ideal chromosome begins—a chromosome that generates a solution with the theoretical optimum number of workstations. If no such chromosome is found, a new population is generated using the genetic operators: parent-child selection (reproduction), crossover, mutation and evaluation. This process is carried out until the termination criteria are met. Figure 6.1 shows how this logical process is followed.

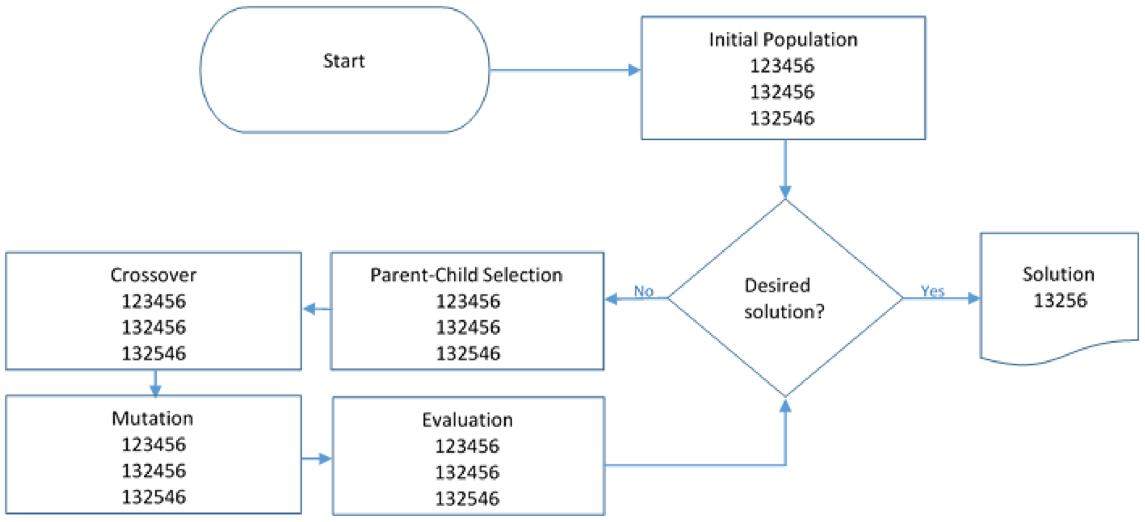


Figure 6.1: Genetic Algorithm Logic

Once the termination criteria are met, the genetic algorithm returns the following exit elements:

- The least number of workstations found given the entry data,
- A list of tasks assigned to each workstation,
- The variation of workloads distributed to workstations, and
- The number of iterations done to achieve the result.

6.1 Codification

The first step in constructing a genetic algorithm is to define a genetic representation, referred to as a codification. The codification scheme used in this research is an integer codification presented by Yow-YuhLeu et al. (1994) for the straight LBP. Each chromosome represents a possible solution to the LBP. Every task is sequentially listed in the order it will be assigned to the workstations—a process known as representation oriented to a sequence. Each gene of the chromosome contains the task number that it represents.

6.2 Initial Population

The initial population is generated randomly. The number of chromosomes is always constant. Many of the possible combinations of genes are irrelevant because they violate the precedence constraints. The method used to generate a valid random sequence of genes is as follows.

Step 1: Select a set of tasks that don’t have precedence tasks and place them in an empty chromosome.

Step 2: Select a task that is available at random (uniform distribution) and add it to the chromosome.

Step 3: Remove from the set of tasks without precedence the selected task. Add all of the immediate successors of the task if all of their predecessors are already in the chromosome.

Step 4: If there are still tasks available, return to Step 2, otherwise, finish the process.

Note that in Step 3, the set of available tasks is refreshed with tasks that satisfy the precedence constraints. By using this method, the original precedence diagram is represented by a linear, isomorphic graph. This way ensures that the initial population of chromosomes is feasible. Figure 6.2 and 6.3 show the precedence diagram for Jackson's Problem and the isomorphic representation used in this research respectively.

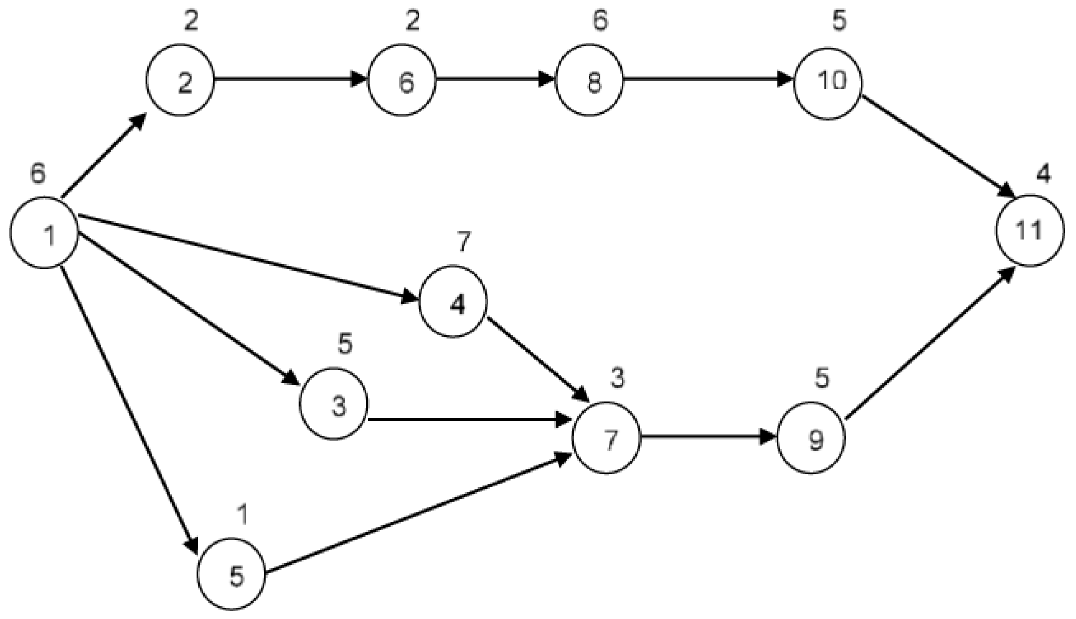


Figure 6.2: Jackson's Problem Precedence Diagram

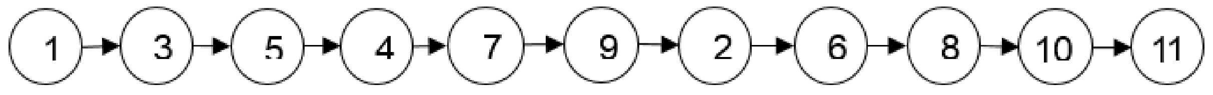


Figure 6.3: Jackson's Problem Isomorphic Representation

This isomorphic diagram makes it easier to see the layout that could be implemented after solving the U-shape LBP. Using the original precedence diagram, it is difficult to visualize how the U-shape assembly line would result but using the generated isomorphic diagram. It is easier to represent it as a U-shape as shown in Figure 6.4.

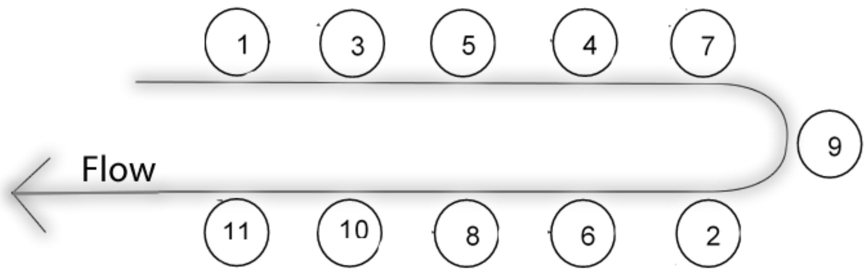


Figure 6.4: U-shaped Isomorphic Diagram

6.3 De-codification

A sequence-oriented representation does not violate the precedence constraints and therefore is denominated feasible sequence. A feasible sequence generates multiple task assignations to the workstations instead of one. It is necessary to appropriately decode each chromosome to assign a single solution to it. This unique solution is the one that gets an aptitude (number of workstations) and smoothness index. The method used to decode the chromosome is described below.

Step 1: Create an empty workstation.

Step 2: If neither the first or last task in the chromosome has a task time less than or equal than the available time, return to Step 1.

Step 3: If only one of the tasks' time is less or equal, assign it to the workstation. Subtract the task time from the available time and eliminate the task from the chromosome. If more than one task could be assigned, select one at random. If the chromosome still has tasks, return to Step 2, if not, end the process.

6.4 Evaluation

Each member of the population is evaluated to determine the probability of it surviving in the next generation or of it being selected for one of the genetic operations. This probability is calculated based on two parameters obtained from each chromosome. The first and most important parameter is determined by the number of workstations resulting from decoding the chromosome. The objective of the first parameter is to minimize the number of workstations (Aptitude). The objective of the second parameter is to minimize the variation of workloads among the workstations (Smoothness Index). According to Yeo Keun Kim (2000), there are several benefits when the assembly line operates in this manner. The production rate is increased.

6.5 Reproduction

The process of reproduction is done starting with a population of ordered chromosomes according to their value of aptitude and suaveness. In this step of the process, the percentage of chromosomes to which the crossover operation will be applied, is selected. The chromosomes that will pass on to the next generation before the mutation process are also selected.

6.6 Crossover

The whole idea of producing new generations of chromosomes through genetic operations is to improve the solutions of those in the initial population.

Basically, crossover is to select two parent chromosomes with the best fitness values and combine their characteristics to produce new child chromosomes. The number of child chromosomes produced is equal to the number of parents involved in the crossover. These child chromosomes also represent a feasible solution. The process of crossover used for this research is presented below.

Suppose two parent chromosomes of equal length have been selected from Jackson’s problem, as shown in Figure 6.5. Two crossover points are selected, represented by the dashed lines.

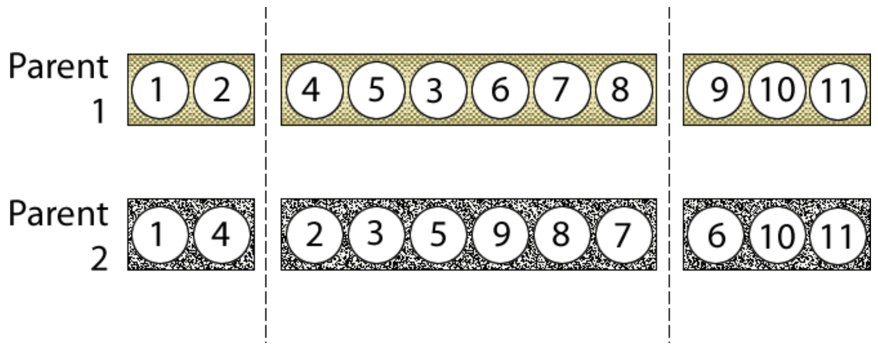


Figure 6.5: Crossover Parents

The middle parts of the two parents must be swapped to create the child chromosomes. However, this is not simple, as feasibility must be maintained. To create the first child, the first and last portion of parent 1 are positioned, leaving all the middle components missing. All of the missing tasks will be placed in the order in which appear in parent chromosome 2. The second child is produced similarly, except the end parts are taken from parent 2 and the middle portion follows parent one’s order. The resulting children are depicted in Figure 6.6.

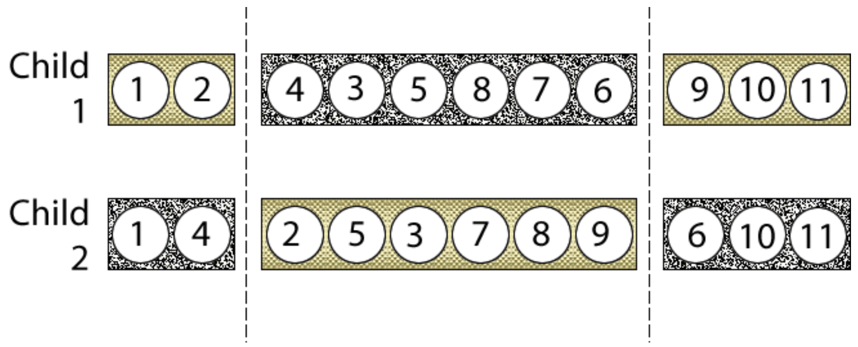


Figure 6.6: Crossover Children

Since both parent chromosomes represented a feasible solution, and the children were produced in a way that no precedence constraints are violated, both resulting chromosomes are feasible, too. It should be acknowledged that these children are similar to their parents in that they preserve the end parts of the chromosome. The purpose of this method is to search for solutions that are similar to the chromosome we have.

6.7 Mutation

Mutations will be applied to every chromosome that was previously selected to mutate. For the algorithm used in this research, a certain type of mutation called mixed mutation was used. The place where a chromosome is mutated is selected randomly. The portion of the chromosome that will not be mutated will be preserved identically. The rest of the chromosome is generated in the same way that the initial population was generated, preserving feasibility. The mutation itself starts at a task that has no precedence and continues until all tasks have been assigned. An example of this is shown in Figure 6.7.

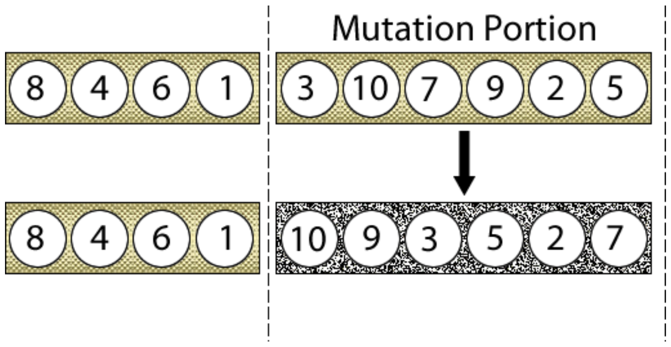


Figure 6.7: Mutation Example

6.8 Termination Criteria

Once a new population is generated, termination criteria are checked. If these aren’t met, another iteration must be done to generate a new population.

There are three criteria used to determine when to stop the genetic algorithm. When any of these three is met, the procedure ends. The first criterion is finding the ideal theoretical solution. The theoretical number of workstations is given by dividing the sum of task times by the cycle time. This is a very rare situation, even using dynamic programming.

The second criterion specifies that if the total number of iterations is reached, no more cycles should start.

The last of the three criteria ends the process if in a certain number of iterations there hasn't been an improvement of at least 1%.

The maximum number of iterations has the objective of balancing the convergence of the algorithm towards an optimum result. This number of iterations depends on the size of the problem to be solved. If there are many tasks in the problem then a greater number of iterations is recommended, otherwise only a few iterations are recommended.

6.9 Parameters

The effectiveness of the results of a genetic algorithm is greatly influenced by the initial parameters. It is important to select these parameters in a way that results in good solutions in a small amount of time. A procedure to select these parameters is presented in Chapter 7.

6.10 Numerical Example

The same problem used in Chapter 5 is used to be solved using a direct-coded genetic algorithm. Once again, the task times, number of tasks and precedence network are preserved. The length of each chromosome will be equal to the amount of tasks this time. So, starting with a population of 20 chromosomes, the population appears as shown in Table 6.1.

Table 6.1 Jackson's Problem Population Using Direct-Coded Genetic Algorithm

Chromosome	
1	1,2,6,4,5,8,3,7,10,9,11
2	1,3,5,4,2,7,6,9,8,10,11
3	1,3,2,5,4,7,6,8,10,9,11
4	1,4,5,3,7,2,6,9,8,10,11
5	1,2,3,6,5,4,7,8,10,9,11
6	1,3,2,6,5,4,8,10,7,9,11
7	1,5,3,2,4,6,8,10,7,9,11
8	1,2,4,3,5,6,7,8,9,10,11
9	1,2,3,6,4,5,7,8,10,9,11
10	1,5,4,3,2,6,7,9,8,10,11
11	1,2,3,4,5,7,6,8,10,9,11
12	1,5,2,6,4,3,8,10,7,9,11
13	1,2,4,6,8,5,10,3,7,9,11
14	1,4,3,2,5,7,6,9,8,10,11
15	1,5,3,2,6,4,7,9,8,10,11
16	1,5,4,3,2,7,9,6,8,10,11
17	1,3,5,2,4,6,8,10,7,9,11
18	1,3,2,4,5,7,9,6,8,10,11
19	1,5,4,2,6,8,3,10,7,9,11
20	1,5,4,3,2,7,9,6,8,10,11

The solution to the U-shaped LBP from chromosome one is shown in Table 6.2.

Table 6.2: Jackson's Problem Solution Using Direct-Coded Genetic Algorithm

Station	V	Will Fit	Assigned	Time left
1	1,11	1,11	1	5
1	2,11	2,11	11	1
2	2,9	2,9	2	9
2	6,9	6,9	9	4
2	6,10	6	6	2
3	4,10	4,10	4	4
3	5,10	5	5	3
4	8,10	8,10	8	5
4	3,10	3,10	3	0
5	7,10	7,10	10	6
5	7	7	7	3

A graphical representation of what this solution would look like the diagram presented in Figure 6.8.

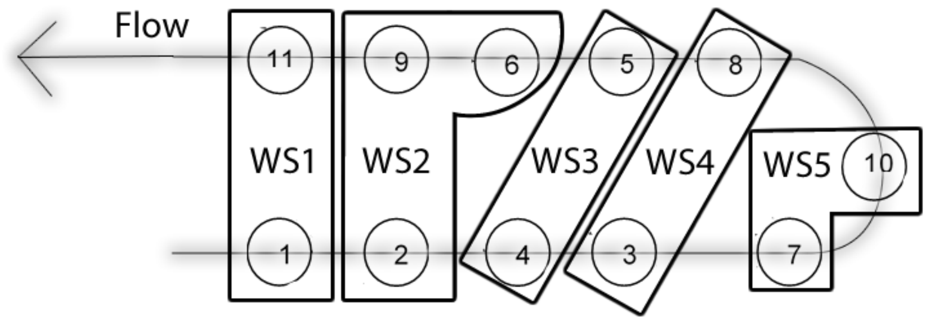


Figure 6.8: Jackson’s Problem Solution Using Direct-Coded Genetic Algorithm

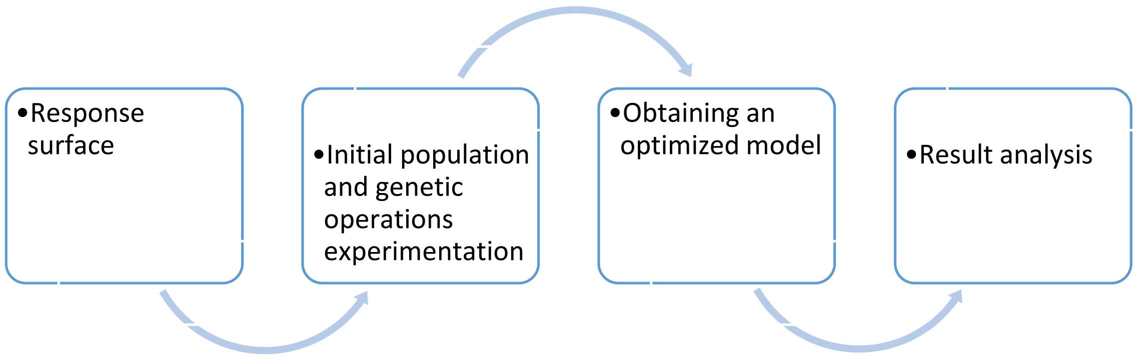
This represents a solution from the initial population. After applying genetic operators, solutions should improve. The best solutions from the initial population (high fitness values) are chosen to reproduce and better child chromosomes are created. This process is repeated until one or more of the termination criteria are met.

This heuristic approach to solving the U-shape LBP is computationally efficient because it searches only for solutions that are very likely to produce good solutions. Even if the best result isn’t achieved, each iteration makes the population ever so slightly better.

CHAPTER 7

EXPERIMENTAL RESULTS

To demonstrate the effectiveness of the direct- and indirect-coded genetic algorithm from previous chapters, a response surface analysis was conducted to obtain the best values for the parameters to be used in the algorithms and reduce the computational time needed to obtain solutions. Using these values, the solutions were compared with the results of known problems taken from the literature. Figure 7.1 shows the process followed to carry out the experimentation.



•Response surface
•Initial population and genetic operations experimentation
•Obtaining an optimized model
•Result analysis

Figure 7.1: Experimentation Process

These results were then compared to the optimal solution (theoretic), as well as solutions obtained using other heuristic methods.

7.1 A Response Surface Model

Response Surface Methods are designs and models for working with continuous treatments when the goal is to find optimum values or to describe the response (Oehlert 2000). The first goal for the Response Surface Method is to find the optimum response. When there are constraints on the design data, the experimental design has to meet requirements of the constraints. The second goal is to understand how the response changes in a given direction by adjusting the design variables

7.1.1 First-Order Response Surface

The relationship between the response variable *y* and independent variables is usually unknown. In general, the low-order polynomial model is used to describe the response surface *f*. A polynomial model is usually a sufficient approximation in a small region of the response surface. Therefore, depending on the approximation of the unknown function *f*, either first-order or second-order models are employed. Furthermore, the approximated function *f* is a first-order model when the response is a linear function of independent variables. A first-order model with *N* experimental runs carried out on *q* design variables and a single response *y* can be expressed as follows [27]:

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_q x_{iq} + \varepsilon_i = \beta_0 + \sum_{j=1}^{q} \beta_{ij} x_{ij} + \varepsilon_i \quad (i = 1,2 \ldots n) \qquad [\ 27\ ]$$

The response *y* is a function of the design variables x_1, $x_2, \ldots, x_q$, plus the experimental error, and the β_j*'s* are regression coefficients. In general, a linear multiple-regression model with a q independent variable takes the form of equation 27.

where $n > q$. The i^{th} observation of the j^{th} independent variable (or factor level) is denoted by x_{ij}. The data structure for the multiple linear regression model is shown in Table 7.1.

Table 7.1: Data for Multiple-Regression Model

y	**x_1**	**x_2**	**…**	**x_q**
y_1	x_{11}	x_{12}	…	x_{1q}
y_2	x_{21}	x_{22}	…	X_{2q}
.	.	.	…	.
.	.	.	…	.
.	.	.	…	.
y_n	x_{n1}	x_{n2}	…	x_{nq}

The multiple-regression model can be written in a matrix form:

$$y = X\beta + e \qquad [\ 28\ ]$$

where:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1q} \\ 1 & x_{21} & x_{22} & \cdots & x_{2q} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n2} & x_{n2} & \cdots & x_{nq} \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_q \end{bmatrix} \quad e = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

y is an *(n X 1)* vector of observations, *X* is an *(n X k)* matrix of values of the independent variables (or factor levels), *ß* is a *(k X 1)* vector of regression coefficients, and *e* is an *(n X 1)* vector of random errors (Montgomery 2005).

If *X* is a *(k X k)* nonsingular matrix, then the linear system *y = Xß + e* has a unique least squares solution given by $\beta = (X'X^{)-1} X' y$. The estimated regression equation is $y = X\beta$.

7.1.2 Fitting Second-Order Model

There are many designs available for fitting a second-order model. The most popular one is the central composite design (CCD). This design was introduced by Box and Wilson. It consists of factorial points (from a 2^q design and 2^{q-k} fractional factorial design), center points and axial points. The following is the representation of 2^q axial points:

x_1	**x_2**	**…**	**x_q**
-a	0	…	0
a	0	…	0
0	-a	…	0
0	a	…	0
…	…	…	…
0	0	…	-a
0	0	…	a

The idea behind a response surface approach is to select the experimental points so that they satisfy some optimality criterion about the model to be used. There are many types of designs that can be used for this purpose. A good strategy is to try with a simple design that has extra degrees of freedom for validation and for checking model adequacy. When a first-order model with center points shows evidence of lack of fit, axial points can be added to the designed experiment to create the CCD. The number of center points n_c at the origin and the distance a of the axial runs from the design center are two parameters in the CCD design. Lack of fit is determined by comparing the response prediction with the response measurements at the center points and this provides information about the curvature of the surface. The additional axial points enable an estimation of the quadratic terms. Figure 7.2 illustrates the graphical view of a central composite design for $q = 2$ factors.

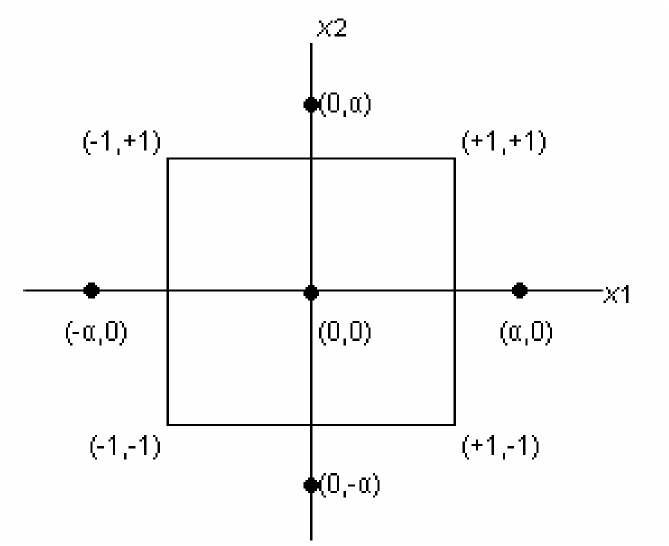


Figure 7.2 Central Composite Design for $q = 2$

7.1.2 Analyzing the Stationary Point

The second-order models can exhibit mathematical features such as minimums, maximums, ridges and saddle points. If an optimum exists, then this point could be a stationary point or a point on the boundary. The stationary point is the combination of design variables where the surface is at either a maximum, a minimum, or a saddle point. If the stationary point is a saddle point, then it is not a local maximum or a minimum. This means that the surface curves upward in some direction and downward in other directions from the stationary point. When the surface is curved in some directions, but is relatively constant in other directions, this type of surface is called ridge system (Oehlert 2000). The stationary point can be found by using matrix algebra. The fitted second-order model in matrix form is as follows:

$$\hat{y} = \hat{\beta}_0 + x'b + x'Bx \quad [\ 29\ ]$$

The derivative of *y* with respect to the elements of the vector *x* is:

$$\frac{\partial \hat{y}}{\partial x} = b + 2Bx = 0 \quad [\ 30\ ]$$

Therefore, the solution to stationary point is:

$$x_s = -\frac{1}{2}B^{-1}b \quad [\ 31\ ]$$

Where:

$$b = \begin{bmatrix} \hat{\beta}_1 \\ \hat{\beta}_2 \\ \vdots \\ \hat{\beta}_q \end{bmatrix} \ and \ B = [\begin{matrix} \hat{\beta}_{11}, & \frac{\hat{\beta}_{12}}{2}, & \dots, & \frac{\hat{\beta}_{1q}}{2} \\ & \hat{\beta}_{22}, & \dots, & \frac{\hat{\beta}_{2q}}{2} \\ & & \ddots & \\ sym. & & & \frac{\hat{\beta}_{qq}}{2} \end{matrix}] \quad [\ 32\ ]$$

b is a *(q X 1)* vector of the first-order regression coefficients and B is a *(q X q)* symmetric matrix whose main diagonal elements are the quadratic coefficients *($\hat{\beta}_{ii}$)* and whose off diagonal elements are one-half the mixed quadratic coefficients *($\beta_{ij}\ i \neq j$)* (Montgomery 2005). In result, the estimated response value at the stationary point can be calculated as:

$$\hat{y} = \widehat{\beta_0} + x_s'b \quad [\ 33\ ]$$

The linear, quadratic and interaction terms in the model were found by the Least Squares Method.

Usually, a Box-Wilson strategy is used for optimizing the response of the model but, due to the nature of our problem, a different approach had to be used. Using the traditional method for continuous variables, a non-integer value can be obtained and that wouldn't be valid since it doesn't make sense to use 2.5 mutations or 15.75 children in the algorithm. This is a problem where there are categorical variables that take on discrete values. However, the categorical variables have a natural ordering in the decision space. Thus, the generation of the response surface was done by treating the variables as if they were continuous variables, but the optimization was performed by exhaustive search. Values, after decoding, corresponding to an integer value were sequentially evaluated using the regression polynomial model found by the least squares method. This way, the minimum computational time and its corresponding optimal solutions for the execution of the two proposed algorithms were found. The coded values that correspond to an integer number for each variable are shown in Appendix D.

7.2 Classification of problems using the genetic algorithm in solved problems from literature.

The 12 LBPs were classified depending on the number of operations, resulting in a total of 61 problems. The three groups, namely A, B and C, contain problems depending on the number of tasks. Group A has problems with a number of tasks ranging between 0-20, group B from 21-50, and group C comprises those that have more than 50 tasks. Then there was a comparison between the ideal theoretical number of stations (N), and the best solution found using a heuristic method (N_{HEU}) according to Ajenbilt (1982). The problems, along with their cycle times and the best solutions found, are shown in Table 7.2.

Table 7.2: Problems With Best Solutions

Problem	Tasks	Cycle Time	Group	N	N_{HEU}
Mertens (1967)	7	6 7 8 10 15 18	A	4.8 4.1 3.6 2.9 1.9 1.6	6 5 5 3 2 2
Bowman (1960)	8	20	A	3.7	5
Jaeschke (1964)	9	6 7 8 10 18	A	6.2 5.3 4.6 3.7 2.1	8 7 6 4 3
Jackson (1956)	11	7 9 10 13 14 21	A	6.6 5.1 4.6 3.5 3.3 2.2	8 6 5 4 4 3
Dar-El (1957)	11	48 62 95	A	3.8 3 2	4 4 3

Table 7.2.: Problems With Best Solutions (cont.)

Mitchell (1957)	21	14 15 21	B	7.5 7 5	8 8 6
Heskiaoff (1968)	28	138 205 216 256 324 342	B	7.4 5 4.7 4 3.2 3	8 6 5 4 4 3
Sawyer (1970)	30	25 26 30 36 41 54 75	B	13 12 10.8 9 7.9 6 4.3	15 14 12 10 9 7 5
Kilbridge (1961)	45	57 79 92 110 138 184	B	9.7 7 6 5.1 4 3	10 8 7 6 5 4
Tounge (1969)	70	176 364 410 468 527	C	19.9 9.6 8.6 7.5 6.7	21 10 9 8 7
Arcus (1963)	83	5048 5853 6842 7571 8412 8898 10816	C	15 12.9 11.1 10 9 8.5 7	16 14 12 11 10 9 8
Arcus (1963)	111	5755 8847 10027 10743 11378 17067	C	26.1 17 15 14 13.2 8.8	27 18 16 15 14 9

7.3 Results for the Indirect-Coded Genetic Algorithm

7.3.1 Analysis of the results using Response Surface Analysis.

Previous research done on solving the LBP using genetic algorithms does not implement any sort of methodology to determine the values for the size of the population, the number of mutations, or the number of crossover operations. They use fixed values for these variables that are arbitrarily selected and kept throughout the algorithm's process. Using an analysis of the response surface, the response time in which solutions are obtained can be used to select values for the variables so that solutions can be obtained in a minimized amount of time. A response surface methodology is widely used in engineering and statistics areas and by implementing it in this experiment, the best levels for the number of chromosomes, number of mutations and number of reproductions to minimize computational time can be selected.

A response surface was performed on four factors and two levels to find the significant factors that affect the average time in which problems are solved in each group using the percentage of children (*x1*), the percentage of mutations (*x2*), the number of chromosomes (*x3*) and the number of genes (*x4*), which are the initial parameters of genetic algorithm. Table 7.3 shows levels used for the four factors.

Table 7.3: Four Factors of Surface Response Model

Factor	Variable	-1	0	1
% of children	x1	20	40	60
% of mutations	x2	10	20	30
# Chromosomes	x3	10	55	100
# Genes	x4	10	16	22

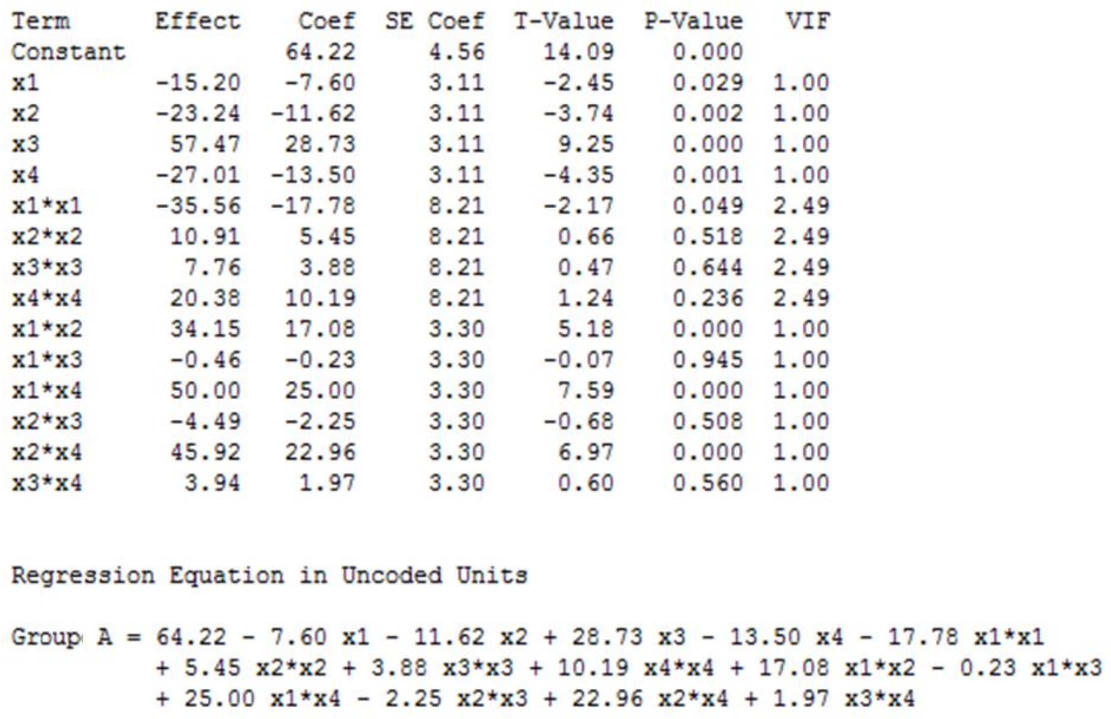
The central composite design was used with a factorial 2^4, eight axial points and four center runs with $\alpha = 1$. A total of 28 runs for each of the problems and 200 generations was conducted. Whenever the best solution was found the execution time was recorded and an average from these times was calculated for each of the three groups. Table 7.4 shows a face-centered central composite design with four center points and the response values for each of the groups (A, B and C).

Table 7.4 Face-Centered Central Composite Design with Center Points and Response Values.

Run	PtType	Blocks	x1	x2	x3	x4	Group A	Group B	Group C
1	1	1	-1	-1	-1	-1	124.157	854.76	737.64
2	1	1	1	-1	-1	-1	34.524	372.57	391.40
3	1	1	-1	1	-1	-1	35.750	514.91	816.60
4	1	1	1	1	-1	-1	13.475	798.70	1251.90
5	1	1	-1	-1	1	-1	199.206	1490.61	2086.76
6	1	1	1	-1	1	-1	85.527	681.94	937.50
7	1	1	-1	1	1	-1	87.795	1143.09	1428.00
8	1	1	1	1	1	-1	42.630	730.95	985.66
9	1	1	-1	-1	-1	1	18.718	336.12	509.04
10	1	1	1	-1	-1	1	3.636	109.85	148.40
11	1	1	-1	1	-1	1	5.815	214.70	202.44
12	1	1	1	1	-1	1	63.882	544.61	762.72
13	1	1	-1	-1	1	1	63.135	1453.95	1678.30
14	1	1	1	-1	1	1	74.151	335.16	420.80
15	1	1	-1	1	1	1	59.163	605.07	701.40
16	1	1	1	1	1	1	134.381	641.62	824.40
17	-1	1	-1	0	0	0	48.830	911.88	1073.27
18	-1	1	1	0	0	0	53.565	699.15	1028.55
19	-1	1	0	-1	0	0	98.940	681.45	570.96
20	-1	1	0	1	0	0	49.918	463.87	729.56
21	-1	1	0	0	-1	0	37.268	287.65	202.86
22	-1	1	0	0	1	0	108.444	508.05	806.78
23	-1	1	0	0	0	-1	100.604	1235.39	1348.27
24	-1	1	0	0	0	1	57.728	863.64	1200.81
25	0	1	0	0	0	0	48.933	876.00	1126.20
26	0	1	0	0	0	0	70.681	700.80	1126.20
27	0	1	0	0	0	0	59.807	876.00	1051.12
28	0	1	0	0	0	0	48.933	700.80	976.04

Table 7.5 shows the results of analyzing the two-level design with center points for group A.

Table 7.5: Group A Complete Model Results



```
Term      Effect    Coef  SE Coef  T-Value  P-Value   VIF
Constant           64.22     4.56    14.09    0.000
x1        -15.20   -7.60     3.11    -2.45    0.029  1.00
x2        -23.24  -11.62     3.11    -3.74    0.002  1.00
x3         57.47   28.73     3.11     9.25    0.000  1.00
x4        -27.01  -13.50     3.11    -4.35    0.001  1.00
x1*x1     -35.56  -17.78     8.21    -2.17    0.049  2.49
x2*x2      10.91    5.45     8.21     0.66    0.518  2.49
x3*x3       7.76    3.88     8.21     0.47    0.644  2.49
x4*x4      20.38   10.19     8.21     1.24    0.236  2.49
x1*x2      34.15   17.08     3.30     5.18    0.000  1.00
x1*x3      -0.46   -0.23     3.30    -0.07    0.945  1.00
x1*x4      50.00   25.00     3.30     7.59    0.000  1.00
x2*x3      -4.49   -2.25     3.30    -0.68    0.508  1.00
x2*x4      45.92   22.96     3.30     6.97    0.000  1.00
x3*x4       3.94    1.97     3.30     0.60    0.560  1.00


Regression Equation in Uncoded Units

Group A = 64.22 - 7.60 x1 - 11.62 x2 + 28.73 x3 - 13.50 x4 - 17.78 x1*x1
          + 5.45 x2*x2 + 3.88 x3*x3 + 10.19 x4*x4 + 17.08 x1*x2 - 0.23 x1*x3
          + 25.00 x1*x4 - 2.25 x2*x3 + 22.96 x2*x4 + 1.97 x3*x4
```

As seen in Table 7.5, all of the P-Values yielded for the main effects indicate that the terms were significant (i.e., less than 0.1).The interactions and squared terms that yielded significant results were x_1 squared, x_1*x_2, x_1*x_4, and x_2*x_4.

To see if the model complies with the assumptions of the residuals being random and normally distributed, both a normal probability plot and a scattered plot is shown. Figures 7.3 and 7.4 show the scattered plot and normal probability plot.

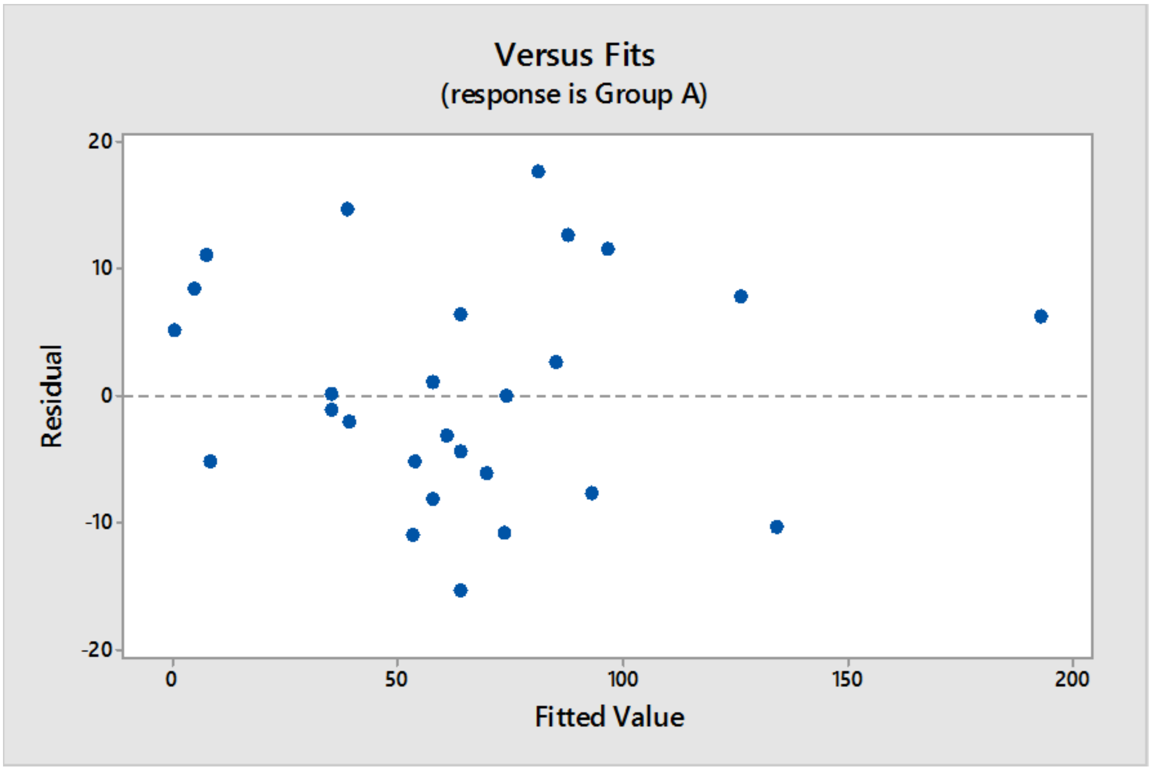


Figure 7.3: Residuals Scattered Plot of Group A

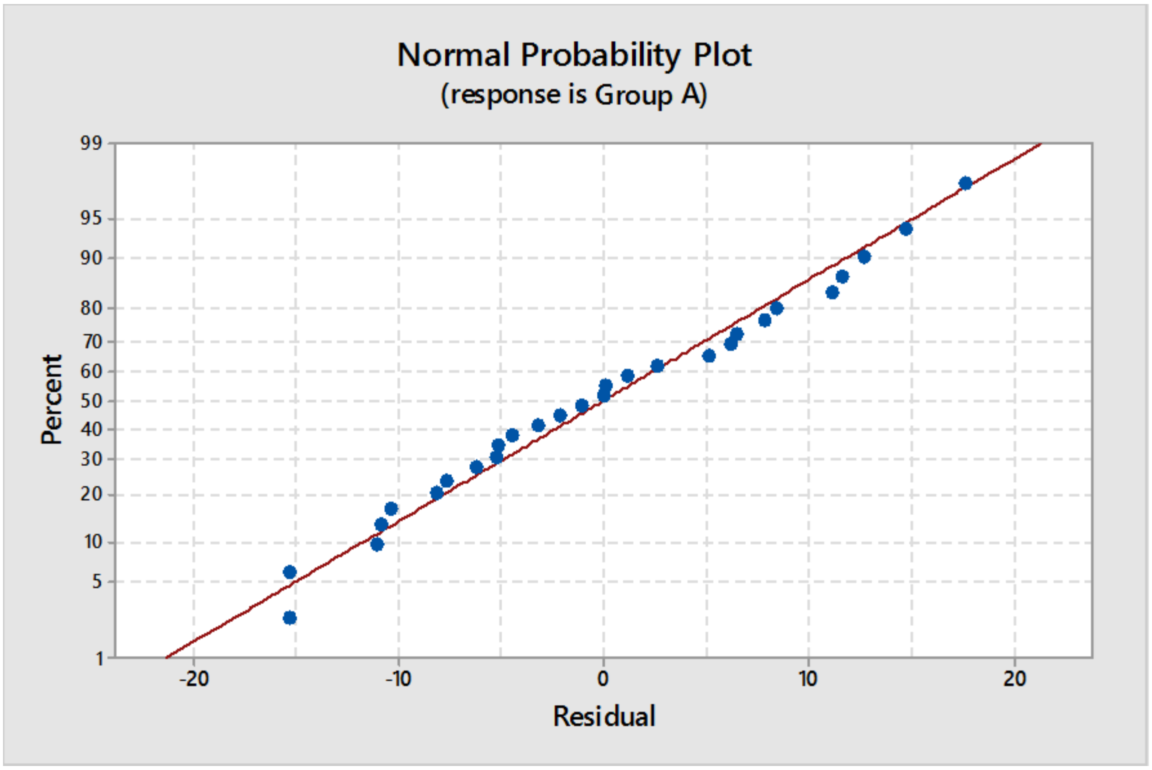
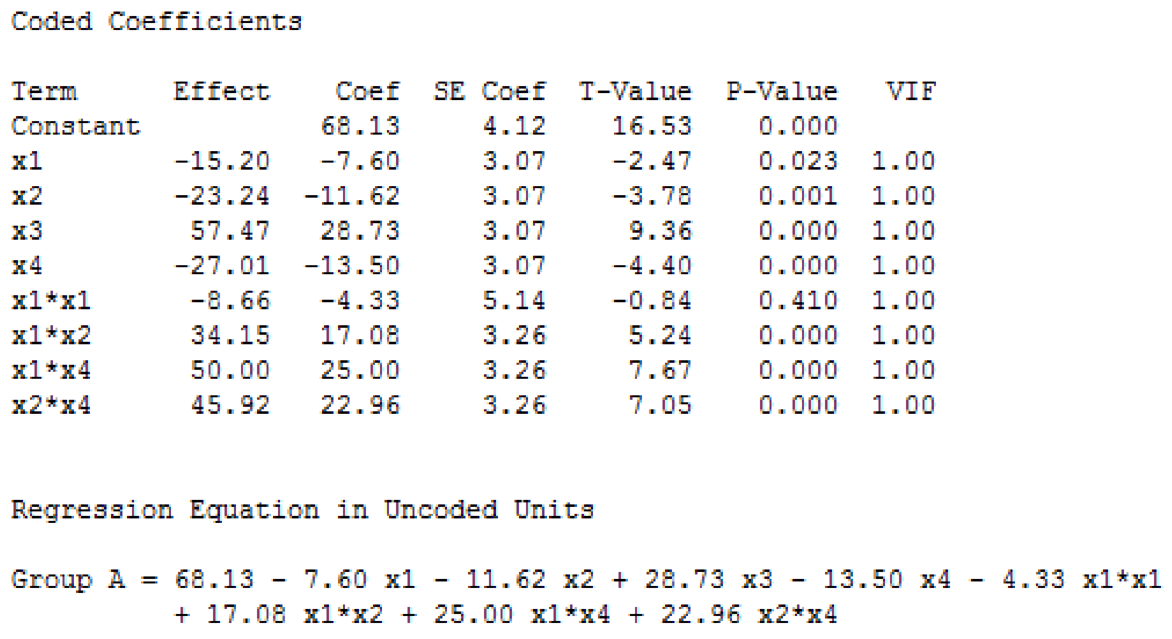


Figure 7.4: Group A Residuals Normal Distribution

If the residuals plot approximately along a straight line, then the normality assumption is satisfied. In Figure 7.4, the residuals can be judged as normally distributed; therefore, the normality assumption is satisfied. Figure 7.3 shows the residuals versus the fitted values. If the regression model is correct and the assumptions are satisfied, the residuals in this plot should be structureless; in particular they should be unrelated to any other variable including the predicted response. (Montgomery 2005). These plots do not show any obvious patterns.

The complete model results (Table 7.3) show that some of the interactions have a large P-value, meaning that their contribution to the model is not significant. A reduced version of the model containing only the significant terms is shown in Table 7.6.

Table 7.6: Group A Reduced Model Results



```
Coded Coefficients

Term      Effect    Coef  SE Coef  T-Value  P-Value   VIF
Constant           68.13     4.12    16.53    0.000
x1        -15.20   -7.60     3.07    -2.47    0.023  1.00
x2        -23.24  -11.62     3.07    -3.78    0.001  1.00
x3         57.47   28.73     3.07     9.36    0.000  1.00
x4        -27.01  -13.50     3.07    -4.40    0.000  1.00
x1*x1      -8.66   -4.33     5.14    -0.84    0.410  1.00
x1*x2      34.15   17.08     3.26     5.24    0.000  1.00
x1*x4      50.00   25.00     3.26     7.67    0.000  1.00
x2*x4      45.92   22.96     3.26     7.05    0.000  1.00


Regression Equation in Uncoded Units

Group A = 68.13 - 7.60 x1 - 11.62 x2 + 28.73 x3 - 13.50 x4 - 4.33 x1*x1
          + 17.08 x1*x2 + 25.00 x1*x4 + 22.96 x2*x4
```

Using the polynomial from the reduced model and an exhaustive search process, the optimum levels of the parameters for group A were found. These values were: 20% children, 30% mutations, 10 chromosomes, and 21 genes.

A similar process was followed to select the levels for the factors of groups B and C. The corresponding tables and figures are shown in Appendix E.

The selected levels for the factors chosen for each problem are summarized in Table 7.7.

Table 7.7: Best Levels for Each Factor

Factor	**Group A**	**Group B**	**Group C**
% of children	20	36.36	40
% of mutations	30	18.18	10
# Chromosomes	10	11	10
# Genes	21	17	19
Estimated Response Values (ms)	1.01	253.80	3.37

7.4 Results for the Direct-Coded Genetic Algorithm

For this section, the problems used were the same ones described in the literature. The 12 problems selected included a variety of task numbers. This results in the same 61 problems used for the indirect coded genetic algorithm in the previous section.

To show the effectiveness of the algorithm, the problems were subjected to experimentation. The first step of this process was to determine the initial values of the parameters for each of the problems. Suitable selection of these parameter values facilitate the attainment of the best solutions.

7.4.1 Determination of initial parameters

The parameters analyzed in this genetic algorithm are number of chromosomes, percentage of children and percentage of mutations. The three mentioned are shown in Table 7.8 below with some initial levels.

Table 7.8: Initial Parameters

Parameter	Number of Levels	Values
Number of Chromosomes	2	10, 100
% of children	2	20, 60
% of mutations	2	10, 30

Each problem is executed during 200 generations and the time it takes for it to reach a solution with the least number of workstations is recorded. This means that the output is the average of time in which the best solution is found.

A response surface of second degree was developed to link the average time it takes to solve the problems from each group given a percentage of children (*x1*), the percentage of mutations (*x2*), and the number of chromosomes (*x3*). These are precisely the variables for the initial parameters and the ones where an optimal value is desired. The objective of the experiment is to minimize the time in which the best solution is met throughout 200 generations. Table 7.9 lists the levels of factors that are coded and used in this experimental design.

Table 7.9: Parameter Levels

Parameter	Variable	Codification		
		-1	0	1
% of children	x_1	20	40	60
% of mutations	x_2	10	20	30
Chromosomes	x_3	10	55	100

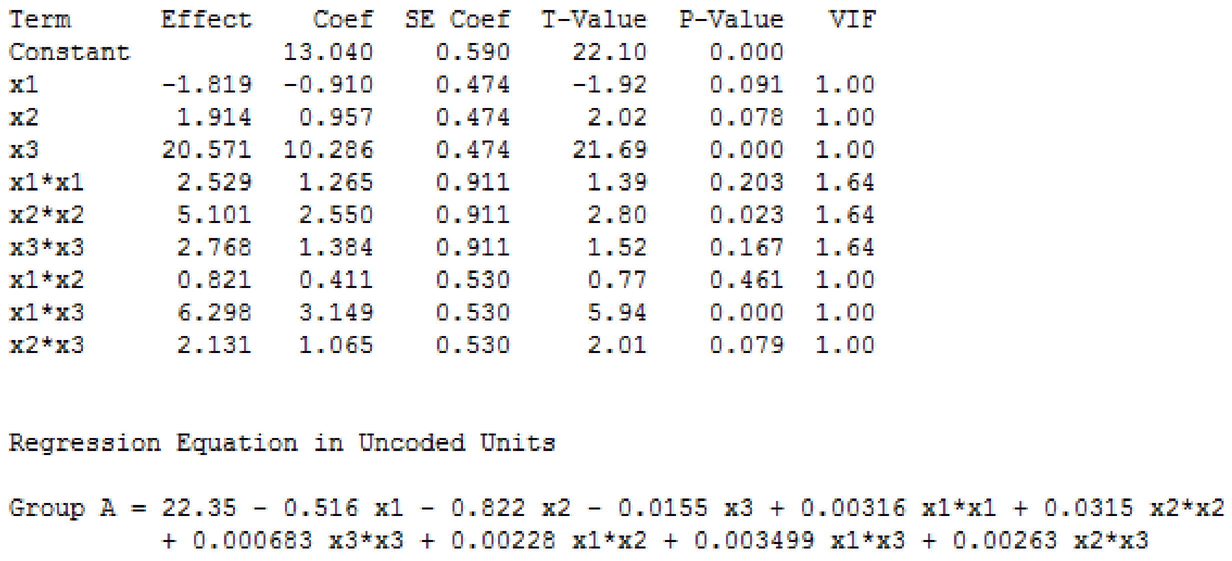
The Central Composite Design is made up by a complete factorial 2^3, four axial points with $\alpha = 1$ for a cubic design and four center point iterations. The values of response for each of the groups and a face-centered central composite design with four center points is shown in Table 7.10.

Table 7.10: Face-Centered Central Composite Design and Response Values

Run	x_1	x_2	x_3	Average time (ms) elapsed		
				Group A	Group B	Group C
1	-1	-1	-1	12.4762	136.07	30.22
2	1	-1	-1	3.4286	155.93	59.33
3	-1	1	-1	12.4762	69.19	68.06
4	1	1	-1	4.3333	81.15	92.22
5	-1	-1	1	24.8571	885.52	981.06
6	1	-1	1	27.6667	1263.78	965.39
7	-1	1	1	28.3810	658.67	130.78
8	1	1	1	33.5714	856.41	246.56
9	-1	0	0	13.6190	308.44	579.94
10	1	0	0	13.7143	271.56	249.61
11	0	-1	0	15.3333	697.56	681.11
12	0	1	0	14.5714	526.67	661.33
13	0	0	-1	3.2381	50.59	53.17
14	0	0	1	24.3333	1427.59	1331.67
15	0	0	0	12.4286	479.56	525.28
16	0	0	0	14.5714	475.81	107.06
17	0	0	0	12.4286	568.37	520.11
18	0	0	0	15.2857	560.30	385.56

An analysis of the response surface design was done for each of the three groups. Figures 7.11-7.12 show the results of the analysis for group A.

Table 7.11: Group A Complete Model Results



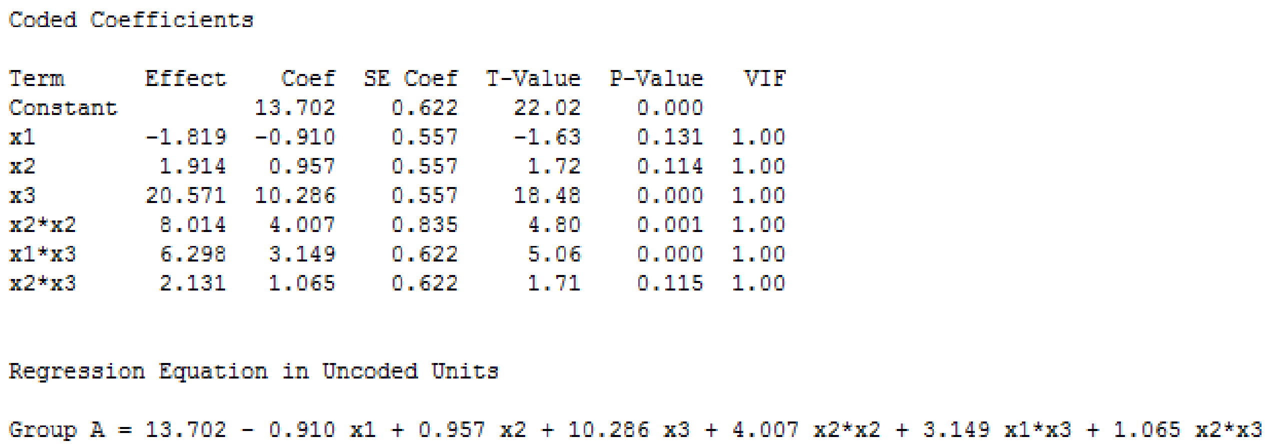
```
Term       Effect    Coef  SE Coef  T-Value  P-Value   VIF
Constant           13.040    0.590    22.10    0.000
x1         -1.819  -0.910    0.474    -1.92    0.091  1.00
x2          1.914   0.957    0.474     2.02    0.078  1.00
x3         20.571  10.286    0.474    21.69    0.000  1.00
x1*x1       2.529   1.265    0.911     1.39    0.203  1.64
x2*x2       5.101   2.550    0.911     2.80    0.023  1.64
x3*x3       2.768   1.384    0.911     1.52    0.167  1.64
x1*x2       0.821   0.411    0.530     0.77    0.461  1.00
x1*x3       6.298   3.149    0.530     5.94    0.000  1.00
x2*x3       2.131   1.065    0.530     2.01    0.079  1.00


Regression Equation in Uncoded Units

Group A = 22.35 - 0.516 x1 - 0.822 x2 - 0.0155 x3 + 0.00316 x1*x1 + 0.0315 x2*x2
          + 0.000683 x3*x3 + 0.00228 x1*x2 + 0.003499 x1*x3 + 0.00263 x2*x3
```

As seen in Table 7.11, the P-Values indicate that the main factors are all significant. The only terms that were not significant in the interactions of factors were x_1 *squared*, x_3 *squared*, and $x_1 * x_2$.

Table 7.12: Group A Reduced Model Results



```
Coded Coefficients

Term      Effect    Coef  SE Coef  T-Value  P-Value   VIF
Constant          13.702    0.622    22.02    0.000
x1        -1.819  -0.910    0.557    -1.63    0.131  1.00
x2         1.914   0.957    0.557     1.72    0.114  1.00
x3        20.571  10.286    0.557    18.48    0.000  1.00
x2*x2      8.014   4.007    0.835     4.80    0.001  1.00
x1*x3      6.298   3.149    0.622     5.06    0.000  1.00
x2*x3      2.131   1.065    0.622     1.71    0.115  1.00


Regression Equation in Uncoded Units

Group A = 13.702 - 0.910 x1 + 0.957 x2 + 10.286 x3 + 4.007 x2*x2 + 3.149 x1*x3 + 1.065 x2*x3
```

The reduced model was optimized to find the minimum response settings. Using the same exhaustive search approach from previous section the best values for the parameters were found. The values found for group A were: 54.54% children, 18.18% mutations, and 11 chromosomes.

A similar process was followed to select the levels for the factors of groups B and C. The corresponding tables and figures are shown in Appendix F.

All the selected levels for groups A, B, and C are summarized in Table 7.13.

Table 7.13: Best Levels for Each Group

Parameter	Group A	Group B	Group C
% of children	54.54	21.42	20
% of mutations	18.18	28.57	20
Number of chromosomes	11	28	10
Estimated Response Value (ms)	0.89	1.33	22.70

7.5 Findings

After preforming all the statistical analysis, the best levels for factors were found. The percentages shown in previous tables equate to an integer value. Table 7.14 shows these values for the direct- and indirect-genetic algorithm.

Table 7.14: Factor Levels Summary

Indirect Coded			
Factor	**Group A**	**Group B**	**Group C**
Number of children	2	4	4
Number of mutations	3	2	1
Number of Chromosomes	10	11	10
Number of Genes	21	17	19
Estimated Response (ms)	1.01	253.80	3.37
Direct Coded			
Number of children	6	6	2
Number of mutations	2	8	2
Number of chromosomes	11	28	10
Estimated Response (ms)	0.89	1.33	22.70

The indirect-coded genetic algorithm must use the following initial parameter levels: for small problems 20%children, 30% mutations, 10 chromosomes and 21 genes. For medium problems: 36.36% children, 18.18% mutations, 11 chromosomes and 18 genes. Large problems required 40% children, 10% mutations, 10 chromosomes and 19 genes.

The parameters that were optimized for the directly coded genetic algorithm were as follows. For small problems, 54.54% children, 18.18% mutations and 11 chromosomes. Medium problems used 21.42% children, 28.57% mutations and 28 chromosomes. For large problems: 20% children, 20% mutations and 10 chromosomes. Using these parameters, each of the algorithms preforms in the least amount of time. Having said that, the minimum time of the direct-coded genetic algorithm is less than the minimum time of the indirect-coded genetic algorithm. This means that the directly coded genetic algorithm obtains solutions faster than the indirectly coded one.

Using these initial parameters, the algorithm was run for each of the problems for all three groups. The results were compared with those results obtained from the literature that used other heuristic or exact approaches. The findings are presented in Table 7.15.

Table 7.15: Problem Solutions

Problem	Cycle time	Best solution		
		Literature solutions	Directly Coded	Indirectly coded
Mertens (1967) 7	6	6	6	6
	7	5	5	5
	8	5	5	5
	10	3	3	3
	15	2	2	2
	18	2	2	2
Bowman (1960) 8	20	5	4	4
Jaeschke (1964) 9	6	8	8	8
	7	7	7	7
	8	6	6	6
	10	4	4	4
	18	3	3	3
Jackson (1956) 11	7	8	7	7
	9	6	6	6
	10	5	5	5
	13	4	4	4
	14	4	4	4
	21	3	3	3
Dar-El (1964) 11	48	4	4	4
	62	4	3	3
	94	3	2	2
Mitchell (1957) 21	14	8	8	8
	15	6	8	8
	21	6	5	5
Heskiaoff (1968) 28	138	8	8	8
	205	6	6	5
	216	5	5	5
	256	4	4	4
	324	4	4	4
	342	3	3	3
Sawyer (1970) 30	25	15	14	14
	27	14	13	13
	30	12	12	11
	36	10	10	10
	41	9	8	8
	54	7	6	6
	75	5	5	5
Kilbridge (1961) 45	57	10	10	10
	79	8	7	7
	92	7	6	6
	110	6	5	5

	138	5	4	4
	184	4	3	3
Tongue (1969) 70	176	21	22	21
	364	10	10	10
	410	9	9	9
	468	8	8	8
	527	7	7	7
Arcus (1963) 83	5048	16	16	16
	5853	14	14	14
	6842	12	12	12
	7571	11	11	11
	8412	10	10	10
	8898	9	9	9
	10816	8	8	8
Arcus (1963) 111	5755	27	28	27
	8847	18	18	18
	10027	16	16	16
	10743	15	15	15
	11378	14	14	14
	17067	9	9	9

As seen in the results presented in this chapter, both the direct- and indirect-genetic algorithms produce good solutions. Both models of the algorithm obtain solutions better than results obtained using other heuristic methods. However, the directly coded method does so at a lower computational cost. This means that even though both algorithms obtain the desired results, a direct-coded algorithm may arrive to the answer faster than the indirect coded algorithm.

7.6 Using the Algorithm in a Case Study Production Line

The genetic algorithms proposed in this research produce good results when comparing them with known results from the literature. To test their performance using problems other than those with known solutions, both the direct- and indirect-genetic algorithms were tested in real production lines.

7.6.1 Indirect-Coded Genetic Algorithm

For implementation of the algorithm it was necessary to have the number of tasks, task time, cycle time, and precedence relationships of the production line. In Figure 7.5, the precedence diagram is shown. Inside the circle is the number of the task outside the circle is the time of the task; inside the box is the cycle time; and the lines represent the precedence relationships that must be met for the specific case of the manufacturing cell of medical products.

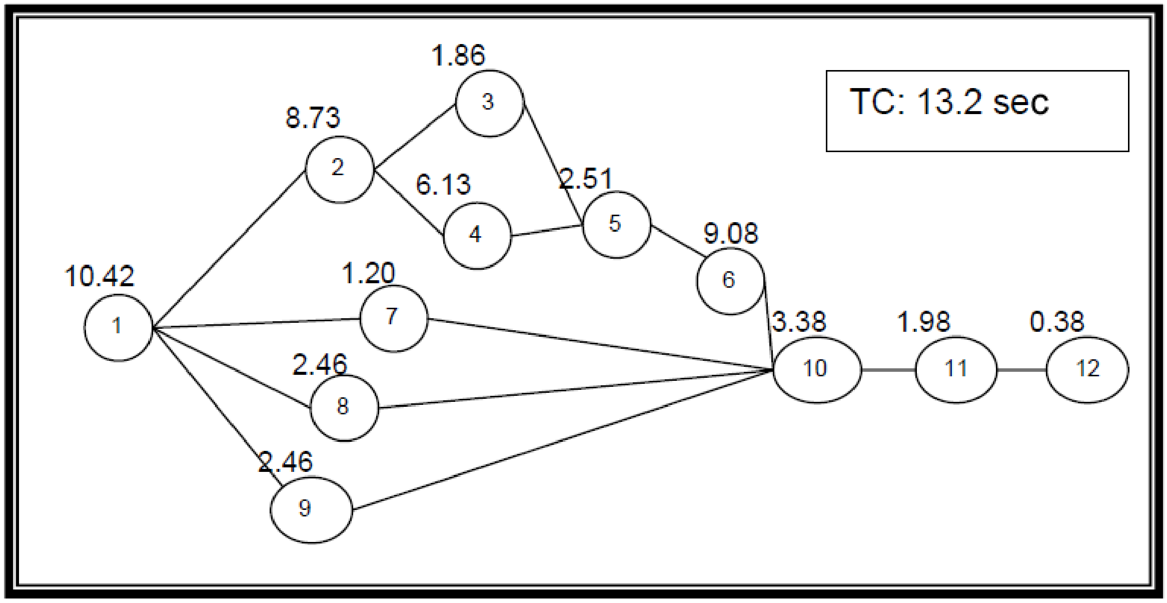


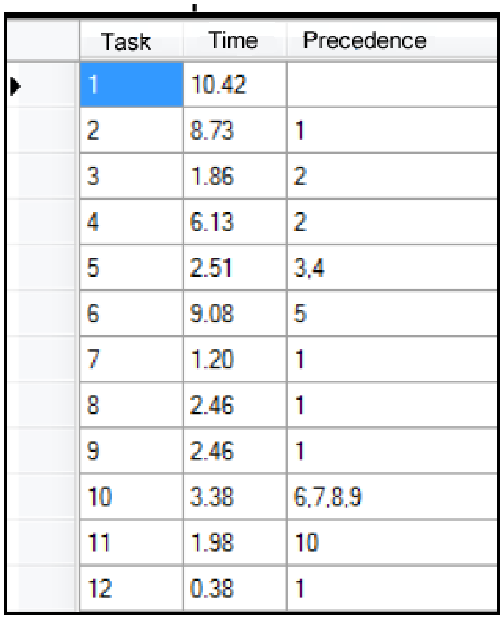
Figure 7.5: Precedence Diagram for Medical Product Problem

Currently, the manufacturing cell consists of five workstations. Sometimes the following situations occur: the output is not constant, there is low production in the first half of the shift and at the end it increases to meet the target goal of the day; downtime is also observed in some stations.

With the implementation of the indirect-coded genetic algorithm in a manufacturing cell, a configuration different than the currently available line was sought to validate if the algorithm is able to find a balancing solution that allows reduction of workstations in the manufacturing cell without exceeding the cycle time while complying with precedence relationships.

The first step of the algorithm is to capture the data in this problem to obtain a table with precedence and cycle time of each task (Table 7.16). The second step is to calculate the weights for each task (Table 7.17). Population is initialized in the program according to the optimal levels of Table 7.14 for group A. Since the problem has only 12 tasks, it is considered a small problem. Table 7.18 shows the ten chromosomes that correspond to the initial population of this problem, with each gene on chromosome representing a heuristic rule. Tasks are then assigned to workstations using the number of rules represented in the genes. If there is a tie, the algorithm moves on to the next gene. If there is a tie throughout all the genes, this chromosome is discarded and the algorithm moves on to the next (see Martinez and Duff (2004) to better understand the process of task allocation). This allocation process is performed by the program.

Table 7.16: Precedence Relations



Task	Time	Precedence
1	10.42	
2	8.73	1
3	1.86	2
4	6.13	2
5	2.51	3,4
6	9.08	5
7	1.20	1
8	2.46	1
9	2.46	1
10	3.38	6,7,8,9
11	1.98	10
12	0.38	1

Table 7.17: Weights for Each Task

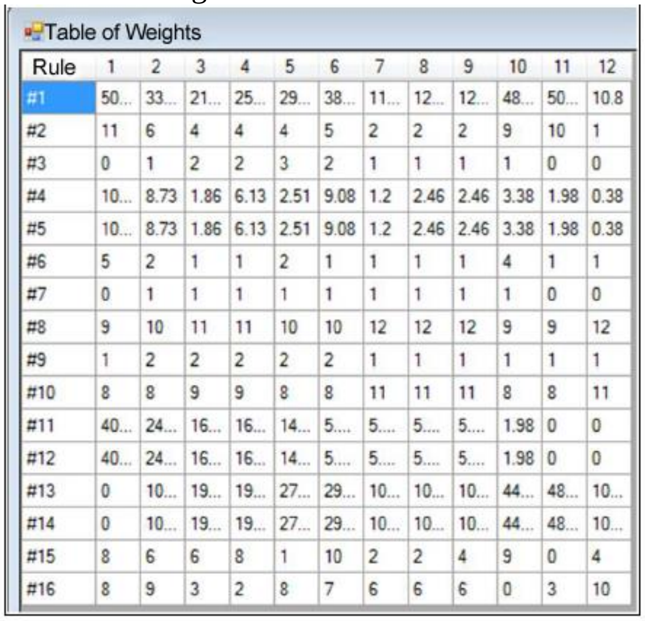
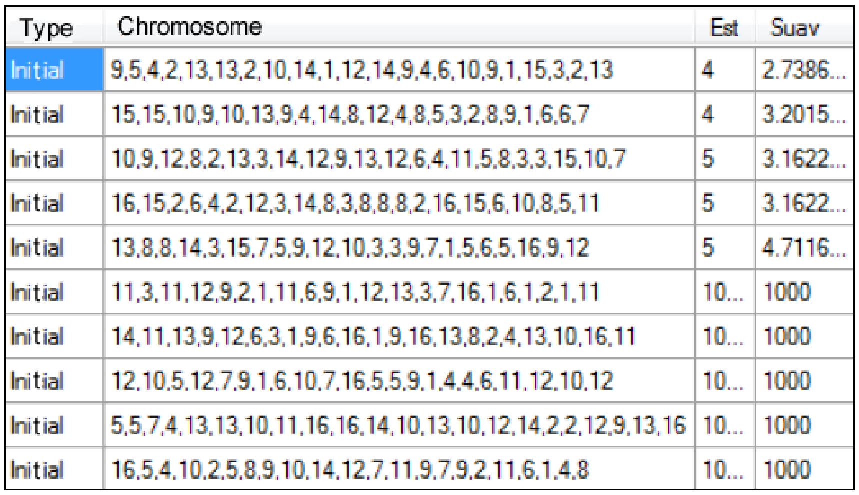


Table of Weights

Rule	1	2	3	4	5	6	7	8	9	10	11	12
#1	50...	33...	21...	25...	29...	38...	11...	12...	12...	48...	50...	10.8
#2	11	6	4	4	4	5	2	2	2	9	10	1
#3	0	1	2	2	3	2	1	1	1	1	0	0
#4	10...	8.73	1.86	6.13	2.51	9.08	1.2	2.46	2.46	3.38	1.98	0.38
#5	10...	8.73	1.86	6.13	2.51	9.08	1.2	2.46	2.46	3.38	1.98	0.38
#6	5	2	1	1	2	1	1	1	1	4	1	1
#7	0	1	1	1	1	1	1	1	1	1	0	0
#8	9	10	11	11	10	10	12	12	12	9	9	12
#9	1	2	2	2	2	2	1	1	1	1	1	1
#10	8	8	9	9	8	8	11	11	11	8	8	11
#11	40...	24...	16...	16...	14...	5....	5....	5....	5....	1.98	0	0
#12	40...	24...	16...	16...	14...	5....	5....	5....	5....	1.98	0	0
#13	0	10...	19...	19...	27...	29...	10...	10...	10...	44...	48...	10...
#14	0	10...	19...	19...	27...	29...	10...	10...	10...	44...	48...	10...
#15	8	6	6	8	1	10	2	2	4	9	0	4
#16	8	9	3	2	8	7	6	6	6	0	3	10

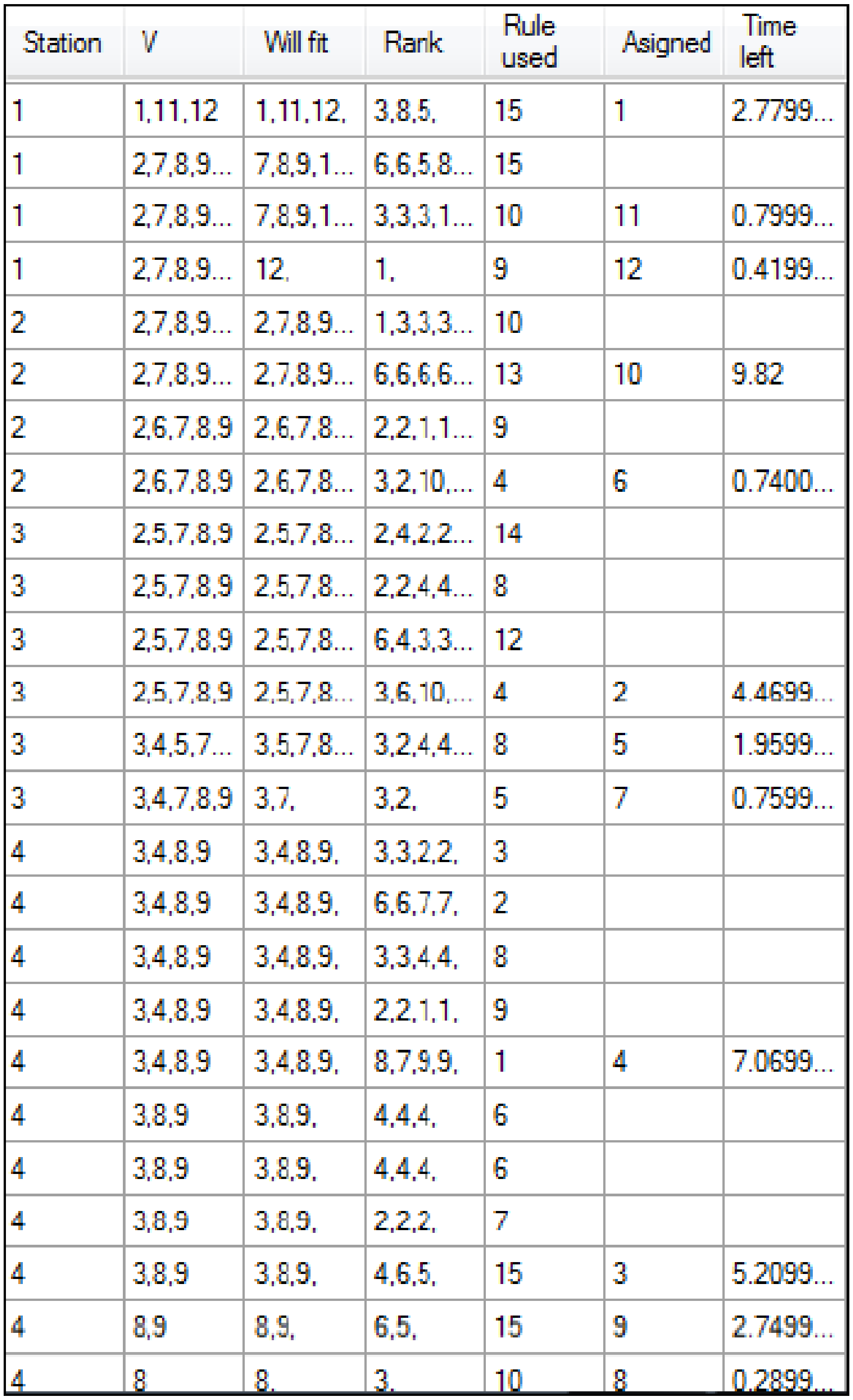
Table 7.18: Generated Chromosomes



Type	Chromosome	Est	Suav
Initial	9,5,4,2,13,13,2,10,14,1,12,14,9,4,6,10,9,1,15,3,2,13	4	2.7386...
Initial	15,15,10,9,10,13,9,4,14,8,12,4,8,5,3,2,8,9,1,6,6,7	4	3.2015...
Initial	10,9,12,8,2,13,3,14,12,9,13,12,6,4,11,5,8,3,3,15,10,7	5	3.1622...
Initial	16,15,2,6,4,2,12,3,14,8,3,8,8,8,2,16,15,6,10,8,5,11	5	3.1622...
Initial	13,8,8,14,3,15,7,5,9,12,10,3,3,9,7,1,5,6,5,16,9,12	5	4.7116...
Initial	11,3,11,12,9,2,1,11,6,9,1,12,13,3,7,16,1,6,1,2,1,11	10...	1000
Initial	14,11,13,9,12,6,3,1,9,6,16,1,9,16,13,8,2,4,13,10,16,11	10...	1000
Initial	12,10,5,12,7,9,1,6,10,7,16,5,5,9,1,4,4,6,11,12,10,12	10...	1000
Initial	5,5,7,4,13,13,10,11,16,16,14,10,13,10,12,14,2,2,12,9,13,16	10...	1000
Initial	16,5,4,10,2,5,8,9,10,14,12,7,11,9,7,9,2,11,6,1,4,8	10...	1000

The algorithm gives different results. Selecting the one having received the smallest number of workstations, in this case four, reveals the allocation of tasks at different workstations as shown in Table 7.19.

Table 7.19: Task Allocation



Station	V	Will fit	Rank	Rule used	Asigned	Time left
1	1,11,12	1,11,12,	3,8,5,	15	1	2.7799...
1	2,7,8,9...	7,8,9,1...	6,6,5,8...	15		
1	2,7,8,9...	7,8,9,1...	3,3,3,1...	10	11	0.7999...
1	2,7,8,9...	12,	1,	9	12	0.4199...
2	2,7,8,9...	2,7,8,9...	1,3,3,3...	10		
2	2,7,8,9...	2,7,8,9...	6,6,6,6...	13	10	9.82
2	2,6,7,8,9	2,6,7,8...	2,2,1,1...	9		
2	2,6,7,8,9	2,6,7,8...	3,2,10,...	4	6	0.7400...
3	2,5,7,8,9	2,5,7,8...	2,4,2,2...	14		
3	2,5,7,8,9	2,5,7,8...	2,2,4,4...	8		
3	2,5,7,8,9	2,5,7,8...	6,4,3,3...	12		
3	2,5,7,8,9	2,5,7,8...	3,6,10,...	4	2	4.4699...
3	3,4,5,7...	3,5,7,8...	3,2,4,4...	8	5	1.9599...
3	3,4,7,8,9	3,7,	3,2,	5	7	0.7599...
4	3,4,8,9	3,4,8,9,	3,3,2,2,	3		
4	3,4,8,9	3,4,8,9,	6,6,7,7,	2		
4	3,4,8,9	3,4,8,9,	3,3,4,4,	8		
4	3,4,8,9	3,4,8,9,	2,2,1,1,	9		
4	3,4,8,9	3,4,8,9,	8,7,9,9,	1	4	7.0699...
4	3,8,9	3,8,9,	4,4,4,	6		
4	3,8,9	3,8,9,	4,4,4,	6		
4	3,8,9	3,8,9,	2,2,2,	7		
4	3,8,9	3,8,9,	4,6,5,	15	3	5.2099...
4	8,9	8,9,	6,5,	15	9	2.7499...
4	8	8,	3,	10	8	0.2899...

According to the results provided by the algorithm, a workstation can be eliminated, setting the cell as shown in Table 7.20. The algorithm was able to reduce from five workstations to only four, as shown in Figure 7.6.

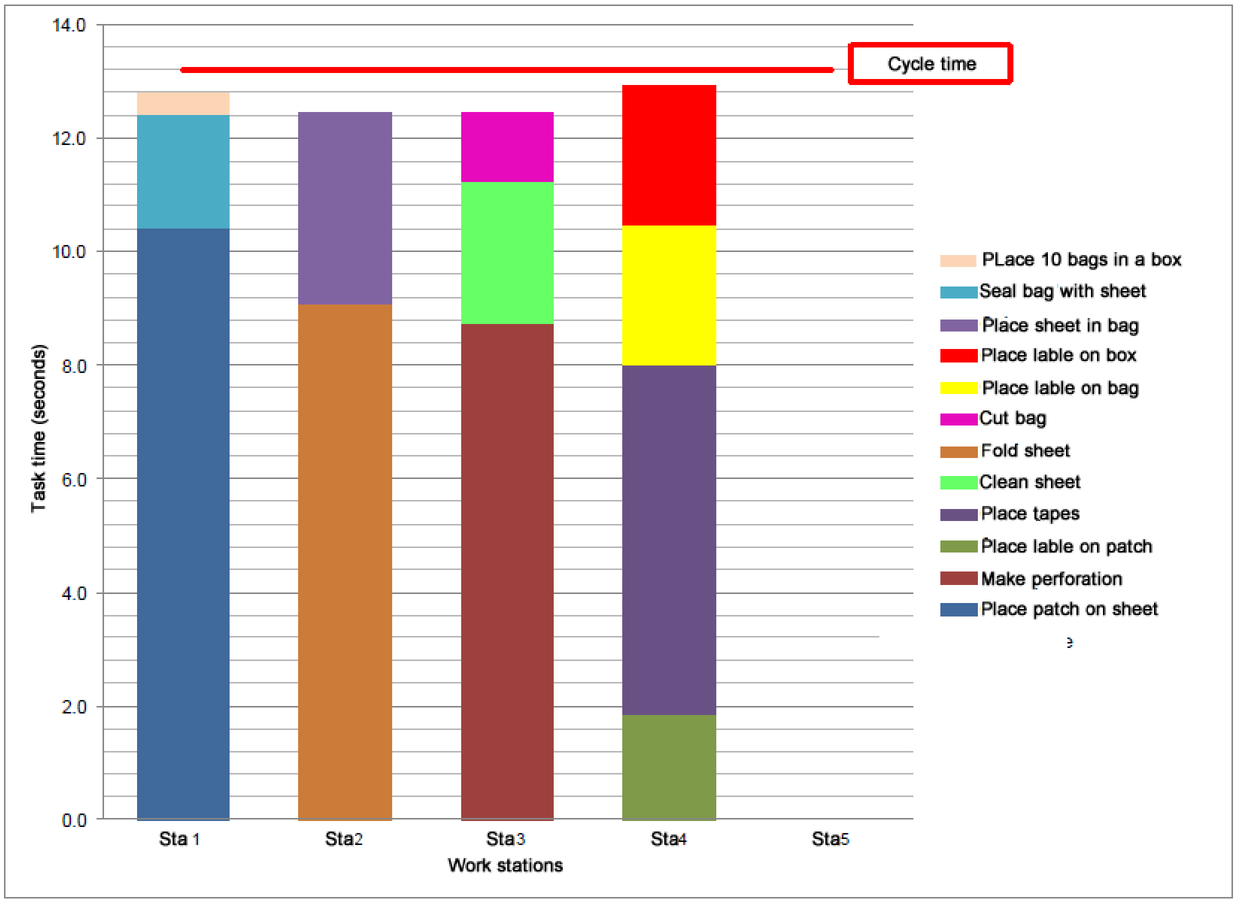


Figure 7.6: Task Time vs. Cycle Time after Algorithm

7.6.2 Directly Coded Genetic Algorithm

To demonstrate the effectiveness of the genetic algorithm, it was applied to a real production line located at an electronics company in Juarez, Chihuahua, Mexico. To implement the directly coded genetic algorithm in this production line, the required information must be gathered: number of tasks, the precedence relations, task times and cycle times. This information is displayed in Figure 7.7.

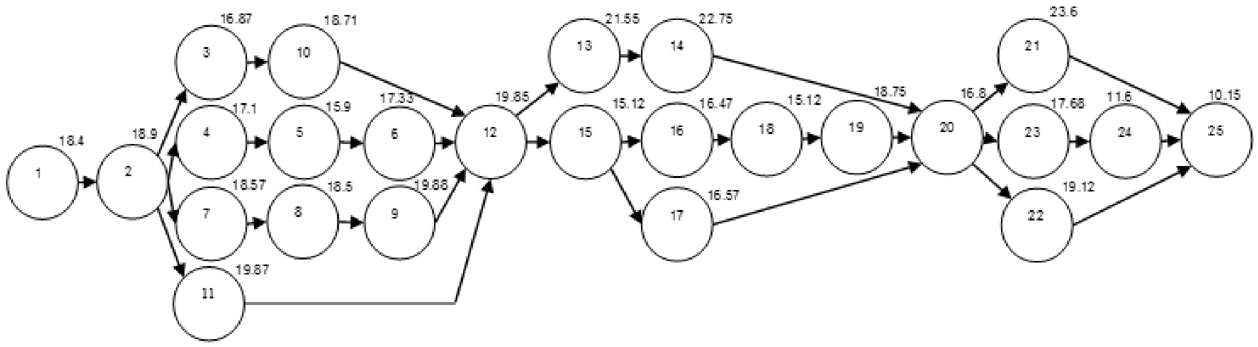


Figure 7.7: Precedence Diagram for Applied Production Line

The cycle time for the production line problem was 40 seconds. As indicated in Formula 5, the theoretical minimum number of workstations is given by dividing the sum of the task times by the cycle time. For this problem, the theoretical number of workstations required is 12. Running the algorithm yields a solution that has precisely 12 workstations. Table 7.20 shows the assignment of tasks results.

Using the cycle time given, the production per shift is 720 finished products. The solution to this LBP was implemented in the production line and the production of the assembly line was recorded along the course of thirty days. The production results from these thirty days are shown in Table 7.21.

A 95% confidence interval was computed, as well as a normal probability plot, to confirm that the production estimated for the given cycle time of 40 seconds is included in this interval (720 parts per shift). The confidence interval obtained was: *(713,19; 724,21),* thus including the 720 of the solution. It can then be concluded that the solution of this LBP produces the expected results.

Table 7.20: Task Assignment for Applied Problem

Station	V	Will Fit	Assigned	Time Left
1	1.25	1.25	25	29.85
1	1.24	1.24	24	18.25
1	1.23	23	23	0.57
2	1.22	1.22	1	21.60
2	2.22	2.22	2	2.70
3	4.22	4.22	22	20.88
3	4.21	4	4	3.78
4	3.21	3.21	3	23.13
4	10.21	10	10	4.42
5	7.21	7.21	7	21.43
5	11.21	11	11	1.56
6	8.21	8.21	8	21.50
6	9.21	9	9	1.62
7	5.21	5.21	21	16.40
7	5.2	5	5	0.51
8	6.2	6.2	20	23.20
8	6.19	6.19	6	5.87
9	12.19	12.19	12	20.15
9	13.19	19	19	1.40
10	13.18	13.18	18	24.88
10	13.16	13.16	13	3.33
11	14.16	14.16	14	17.25
11	15.16	15.16	15	2.13
12	17.16	17.16	17	23.43
12	16	16	16	6.96

Table 7.21: Production During 30 Days

Day	Production
1	714
2	716
3	694
4	727
5	721
6	726
7	739
8	735
9	749
10	695
11	738
12	709
13	735
14	744
15	695
16	697
17	716
18	711
19	719
20	722
21	740
22	718
23	720
24	713
25	725
26	711
27	707
28	697
29	703
30	725

CHAPTER 8

SUMMARY AND CONCLUSIONS

A direct- and an indirect-coded genetic algorithm are developed for solving U-shaped LBPs. In the case of the indirect-coded genetic algorithm, sixteen heuristic rules were taken from the traditional straight line LBP. However, these rules had never been used in a U-shape model. This research introduces a modified version so that they could now be used in the U-shape LBP.

Each heuristic rule was represented as a gene in the chromosomes for tie-breaking results during the task assigning process. For the directly coded algorithm, each chromosome represents a solution to the LBP and the genes represent the actual tasks to assign. To construct this chromosome, a new method is developed that consists of creating a diagram that is isomorphic to the original precedence diagram. Also, crossover and mutation operations are conducted in a way that precedence relations are not violated.

In this research, the two genetic algorithms are tested using 61 problems taken from literature review. The solutions that were obtained from the existing methods in the literature review were compared to the results obtained by using the two developed genetic algorithms. The goals were to increase efficiency by obtaining a solution with the least number of workstations possible, to get the solution in the least amount of time, and to have a minimum smoothness index for the solution. The indirectly coded genetic algorithm obtained better solutions than the known solutions from literature in 26% of the cases; it obtained an equivalent solution in 62% of the cases (not better, not worse); and a worse solution the remaining 12%. The directly coded genetic algorithm obtained better solutions than the known solutions from literature in 22% of the cases;

72% of the problems had a solution that was equivalent; and 6% of the time it generated a solution with more workstations than the solutions from literature The directly coded method obtains a solution faster than the indirectly coded method.

The experimental results of the previous chapters show that the proposed genetic algorithms solve the U-shaped LBP by effectively finding a desired amount of workstations within an acceptable execution time.

The necessity of obtaining a solution within an acceptable computational time is met. In the vast majority of the solved problems, less than a second was needed to arrive with the answer. An optimal solution was not found in every case, but in all cases a very good solution was achieved in a very small amount of time.

Another advantage of these algorithms is that they do not attain just a single solution, they arrive at various solutions with the same number of workstations. This can be especially useful in a case where a particular solution might not be applicable due to external factors independent from precedence constraints and cycle times or where there are additional objective or subjective factors that were not considered.

Knowing what the algorithms are capable of achieving, they were implemented in a real industrial scenario. The two genetic algorithms were able to improve production line balance in these real-world problems. This demonstrates their successful use in more realistic industry problems beyond the theoretical problems from literature review of this research.

Though the results obtained fulfilled the scope of research expectations, it will still be valuable to apply the approaches developed to a broader spectrum of problems—Those having larger sizes, more tasks, and more complex precedence relations. Due to the lack of larger problems with known results in the literature, it was not possible in the current research to test the algorithms with larger and more complex problems. Future researchers should also experiment with more real-world industry problems.

Additionally, a hybrid approach with the two algorithms could be implemented. The first set of solutions from the indirect-coded genetic algorithm can be used as the first initial population for the directly coded genetic algorithm. This first population must be better than the one generated at random as used in the current version of the direct-coded genetic algorithm. This is possible due to the nature of the heuristic rules employed in the indirectly coded genetic algorithm. It is known that the first generation obtained by this algorithm leads to good results. The improvement of that first generation will result in a better and more productive output of solutions.

REFERENCES

Aase Gerald R. (1999) Algorithms for Increasing Labor Productivity in U-shaped Assembly Systems, PhD Dissertation, Indiana University.

Ajenblit D. A. (1992) Applying Genetic Algorithms to the U-shaped Assembly Line Balancing Problem, Proceedings of the IEEE Conference on Evolutionary Computation, pp. 96-101.

Avikal S, Jain R, Mishra PK, Yadav HC (2013) A Heuristic Approach for U-Shaped Assembly Line Balancing to Improve Labor Productivity. ComputIndEng, v 64, pp. 895–901.

Baybars (1984) A Survey of Inexact Algorithms for the Simple Assembly Line Balancing Problem, GSIA, Carnegie-Mellon University, pp. 82-83.

Baybars (1986) An Efficient Heuristic Method for the Simple Assembly Line Balancing Problem, International Journal of Production Research, v 24.

Baykasoglu A, Ozbakir L (2007) Stochastic U-line balancing using genetic algorithms, Int J Adv Manuf Technol, v 32, pp. 139-147.

Berger, I., Bourjolly, J.-M., Laporte, G., (1992) Branch-and-bound algorithms for the multi-product assembly line balancing problem, European Research, v 58, pp. 215–222.

Bowman E. H. (1960) Assembly Line Balancing by Linear Programming, Operations Research, v 8, pp. 385-389.

Boysen N, Fliedner M (2008) A versatile algorithm for assembly line balancing, Eur J Oper Res, v 184, pp. 39-56.

Cakir B, Altiparmak F, Dengiz B (2011) Multi-objective optimization of a stochastic assembly line balancing: a hybrid simulated annealing algorithm, Comput Ind Eng, v 60, pp. 376-384.

Chung-Hung C., Angappa G., Kin-Chuen H. (2010) Improved dynamic programming model implementation for balancing the U-shaped production line, International Journal of Industrial and Systems Engineering, v 6, pp. 463-482.

Cuoglu IS, Erel E, Alp A (2009) Ant Colony Optimization for the Single Model U-Type Assembly Line Balancing Problem, Int. J Prod Econ, v 120, pp. 287–300

Easton, F., Faaland, B., Klastorin, T.D., Schmitt, T., (1989) Improved network based algorithms for the assembly line balancing problem, International Journal of Production Research, v 27, pp. 1901-1915.

Gokcen H, Agpak K (2006) A Goal Programming Approach to Simple U-line Balancing Problem, Eur J Oper Res, v 171, pp. 577–585.

Gokcen H, Agpak K, Gencer C, Kizilkaya E (2005) A Shortest Route Formulation of Simple U-Type Assembly Line Balancing Problem, Appl Math Model, v 29, pp. 373–380.

Gutjar AL, Nemhauser GL (1964) An Algorithm for the Line Balancing Problem. Manag Sci; v 11-2, pp 308-315.

Held, M., Karp R.M., Shareshian, R. (1963) Assembly line balancing – Dynamic programming with precedence constraints, Operations Research, v 11, pp. 442-459.

Hoffmann, T.R., (1993) Response to note on microcomputer performance of ''FABLE'' on Hoffmann's data sets, Management Science, v 39, pp. 1192–1193.

Jackson J.R., (1956) A Computing Procedure for a Line Balancing Problem, Management Sci., v 2-3, pp 261-272.

Jayaswal and Agarwal (2014) Balancing U-Shaped Assembly Lines with Resource Dependent Task Times: A Simulated Annealing Approach, Journal of Manufacturing Systems, v 33-4, p 522-534.

Johnson, R.V., (1993) Note: Microcomputer performance of ''FABLE'' on Hoffmann's data sets, Management Science, v 39, pp. 1190–1193.

Li-yun XU, Loï C, Wei LIU, Qi-fang GU, Ai-ping LI (2014) Improved Computer Methods for Sequencing Operations for U-shaped Assembly Lines, International Conference on Industrial Engineering Management, pp. 51.

Liu SB, Ong HL, Huang HC (2005) A bidirectional heuristic for stochastic assembly line balancing type-II problem, Int J Adv Manuf Technol, v 25, pp. 71-77.

Manavizadeh, N, Rabbani, M, & Radmehr, F (2015) A New Multy-Objective in Order to Balancing and Sequencing U-Shaped Mixed Model Assembly Line Problem: A Proposed Heuristic Algorithm, International Journal of Advaced Manufacturing and Technology.

Martinez, U, Duff, W.S. (2004). Heuristic Approaches to Solve the U-Shaped Line Balancing Problem Augmented by Genetic Algorithms. Proceedings of the 2004 Systems and Information Engineering, Design Symposium, pp 287-293.

Milas Gene H., (1990) Assembly Line Balancing...Let's Remove The Mystery, Industrial Engineering, v 22-5, pp. 31-36.

Miltenburg G. J., Wijngaard J., (1994) The U-Line Balancing Problem, Management Science, Vol. 40-10, pp. 1378-1388.

Miralles C, Garcia-Sabater JP, Andres C, and Cardos M, (2008) Branch and bound procedures for solving the Assembly Line Worker Assignment and Balancing Problem: Application to Sheltered Work Centers for Disabled, Discrete Applied Mathematics, v 156-3, pp 352-367.

Montgomery, Douglas C. (2005) Design and Analysis of Experiments: Response surface method and Designs, New Jersey: John Wiley and Sons, Inc.

Oehlert, Gary W. (2000) Design and Analysis of Experiments: Response Surface Design, New York: W.H. Freeman and Company.

Organ D. & Azizoglu M. (2015) A branch and bound method for the line balancing problem in U-shaped assembly lines with equipment requirements, Journal of Manufacturing Systems, v 36, pp. 46-54.

Ozcan U, Toklu B (2009) A New Hybrid Improvement Heuristic Approach to Simple Straight And U-Type Assembly Line Balancing Problems, J IntellManuf, v 20, pp. 123–136.

Ozcan U (2010) Balancing stochastic two-sided assembly lines: a chance-constrained peicewise-linear, mixed integer program and a simulated annealing algorithm, Eur J Oper Res, v 205, pp. 81-97.

P. Mertens. (1967) Assembly Line Balancing By Partial Enumeration. Ablauf- und Planung-forschung, v 8.

Ponnambalam, S.G. Aravindan, P. and Naidu, G. M. (2000), Multi-Objective Genetic Algorithm for Solving Assembly Line Balancing Problem, International Journal of Advanced Manufacturing Technology, v 16-5, pp. 341-352

Schrage, L., Baker, K.R., (1978) Dynamic programming solution of sequencing problems with precedence constraints, Operations Research, v 26, pp. 444-449.

Sivasankaran P. and Shahabudeen P. (2014) Literature Review of Assembly Line Balancing Problems, The International Journal of Advanced Manufacturing Technology; v 73, Issue 9-12, pp 1665-1694.

Sprecher, A., (2003) Dynamic search tree decomposition for balancing assembly lines by parallel search, International Journal of Production Research, v 41, pp. 1413–1430.

Y. Leu, L. Matheson, L. Rees. (1994) Assembly Line Balancing Using Genetic Algorithms with Heuristic-Generated Initial Populations and Multiple Evaluation Criteria. Decision Sciences, vol. 25-4.

Yegul MF, Agpak K, Yavuz M (2010) A New Algorithm for U-Shaped Two-Sided Assembly Line Balancing. Trans Can SocMechEng, v 34-2, pp. 225–241.

Zhang Z, Cheng W, Cheng Y and Liang J (2008) A Novel Ant Colony Optimization Algorithm for U-Shaped Line Balancing Problem, ICNC, Fourth International Conference on Natural Computation, v 7, pp. 455–459.

BIBLIOGRAPHY

Anderson, E., & Ferris, M. (1994). Genetic Algorithms for Combinational Optimization: The Assembly Line Balancing Problem, ORSA Journal on computing, pp. 161-173.

Andres, C., Miralles, C., & Pastor, R. (2006). Balancing and Sequencing Tasks in Assembly Lines With Sequence-Dependent Setup Times, *European Journal of Operational Research*.

Arcus, A. (1963). An Analysis of a Computer Method of Sequencing Assembly Line Operations. Berkley: University of California.

Bautista, J., & Pereira, J. (2002). Ant algorithms for Assembly Line Balancing. Lecture notes in Computer Science 2463, pp. 65-75.

Becker, C., & Scholl, A. (2004). A Survey on Problems and Methods in Generalized Assembly Line Balancing. European Journal of Operations Research.

Bolat, A. (1997). Stochastic Procedures for Scheduling Minimum Job Sets on Mixed-Model Assembly Lines. The Journal of the Operational Research Society 48, 490-501.

Boysen, N., Fliedner, M., & Scholl, A. (2006). A Classification of Assembly Line Balancing Problems. Jena: Jenaer Schriften zur Wirtschaftswissenschaft 12.

Brian, T., Patterson, J., & Gehrlein, W. (1985). An Integer Programming Algorithm with Network Cuts for Solving the Assembly Line Balancing Problem. Management Science 30, pp. 85-99.

Capacho L. (2007). ASALBP: The Alternative Subgraphs Assembly Line Balancing Problem. Formalization and Resolution Procedures. (Dissertation, Technical University of Catalonia)

Chen, S. (2003). Just-In-Time U-shaped Assembly Line balancing. Bethlehem: Lehigh University.

Chiang, W. (1998). The Application of a Tabu Search Metaheuristic to the Assembly Line Balancing Problem. Annals of Operations Research, pp. 209-227.

Corominas A., Pastor R. & Plans J. (2008). Balancing Assembly Line with Skilled and Unskilled Workers. Omega, 36(6), pp. 1126-1132.

Elsayed, E., & Boucher, T. (1994). Analysis and Control of Production Systems. New Jersey: Prentice Hall International Series in Industrial and Systems Engineering.

Falkenauer, E. (2005). Line Balancing in the Real World. En A. Bouras, B. Gurumoorthy, & R. Sudarsan, Product lifecycle management, pp.360-370.

Feo, T., Resende, M., & Smith, S. (1994). A Greedy Randomized Adaptive Search Procedure for Maximum. Operations Research 42, pp. 860-878.

Ford, H. (1922). My Life and Work. Garden City: Doubleday Page & Co.

Freeman, J., & Jucker, J. (1963). The Line Balancing Problem. Journal of Industrial Engineering 18, pp. 551-562.

Ghosh, S., & Gagnon, R. (1989). A Comprehensive Literature Review and Analysis of the Design, Balancing and Scheduling of Assembly Systems. International Journal of Production Research 27, pp. 637-670.

Glover, F. (1989). Tabu Search - Part I. ORSA J. Comput., pp. 190-206.

Goldberg, D.E., "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison-Wesley, USA, 1989.

Gonçalves, J., & Raimundo de Almeida, J. (2002). A Hybrid Genetic Algorithm for Assembly Line Balancing. Journal of Heuristics 8, pp. 629-642.

Guerriero, F., & Miltenburg, J. (2003). The Stochastic U-line Balancing Problem. Naval Research Logistic 50, pp. 31-57.

Hackman, S., Magazine, M., & Wee, T. (1989). Fast, Effective Algorithms for Simple Assembly Line Balancing Problems. Operations Research 37, pp. 916-924.

Helgeson, W., & Birnie, D. (1961). Assembly Line Balancing Using the Ranked Positional Weighting Technique. Journal of Industrial Engineering 12, pp. 394-398.

Holland, J. (1975). Adaptation in Natural and Artificial Systems. Ann Arbor: University of Michigan Press.

Karp, R. (1972). Reducibility Among Combinational Problems. Complexity of Computer Applications. New York: Plenum.

Kildbridge, M., & Wester, L. (1961). A Heuristic Method of Assembly Line Balancing. Journal of Industrial Engineering 12, pp. 292-298.

Kim, Y. K., Kim, S. J., & Kim, J. Y. (2000). Balancing and Sequencing Mixed-Model U-lines with a Co-Evolutionary Algorithm. Production Planning & Control: The Management of Operations Volume 11, Issue 8, pp. 754-764.

Klein, M. (1963). On Assembly Line Balancing. Operations Research 11, 274-281.

Leu, Y.-Y., Matheson, L., & Rees, L. (1994). Assembly Line Balancing Using Genetic Algorithms with Heuristic-Generated Initial Populations and Multiple Evaluation Criteria. Decision Sciences 25, pp. 581-605.

Lindroth P. & Patricksson M. (2011) Pure Categorical Optimization- a Global Descent Approach, Chalmers University of Technology.

McMullen, P., & Freizer, G. (1998). Using Simulated Annealing to Solve a Multi-Objective Assembly Line Balancing Problem with Parallel Workstations. International Journal of Production Research 36, pp. 2717-2741.

McMullen, P., & Tarasewich, P. (2003). Using Ant Techniques to Solve the Assembly Line Balancing Problem. IIE Transactions 35, pp. 605-617.

Mendes, A., Ramos, A., Simaria, A., & Vilarinho P. (2005). Combining Heuristic Procedures and Simulation Models for Balancing a PC Camera Assembly Line. Computers & Industrial Engineering 49, pp. 413-431.

Mertens, P. (1967). Assembly Line Balancing by Partial Enumeration. Ablauf- und Planung-forschung 8.

Monden, Y. (1998). Toyota Production System - An Integrated Approach to Just-in-Time. Dordrecht: Kluwer.

Resende, M., & Ribeiro, C. (2003). Greedy Randomized Adaptive Search Procedures - Handbook of Metaheuristics. International Series in Operations Research & Management Science 57, pp. 219-249.

Rubinovitz, J., & Levitin, G. (1995). Genetic Algorithm for Assembly Line Balancing. International Journal of Production Economics, pp. 343-354.

Salveson, M. (1955). The Assembly Line Balancing Problem. Journal of Industrial Engineering 6, pp. 18-25.

Scholl, A., & Becker, C. (2004). State-of-the-Art Exact and Heuristic Solution Procedures for Simple Assembly Line Balancing. European Journal of Operational Research 168, pp. 666-693.

Scholl, A., & Klein, R. (1999). UNLINO: Optimally Balancing U-shaped JIT lines. International Journal of Production Research 37, pp. 721-736.

Scholl, A., & Voß, S. (1996). Simple Assembly Line Balancing - Heuristic Approaches. Journal of Heuristics 2, pp. 217-244.

Suresh, G., & Sahu, S. (1994). Stochastic Assembly Line Balancing Using Simulated Annealing. International Journal of Production Research 32, pp. 1801-1810.

Talbot, F., & Patterson, J. (1981). A Comparative Evaluation of Heuristic Line Balancing Techniques. Ann Arbor: University of Michigan.

Tonge, F. (1969). Summary of a Heuristic Line Balancing Procedure. Management Science 7, 21-42.

Urban, T. (1998). Optimal Balancing of U-shaped Assembly Lines. Management Science 44, pp. 738-741.

Van Assche, F., & Herroelen, W. (1979). Optimal Procedure for the Single-model Deterministic Assembly Line Balancing Problem. European Journal of Operational Research 3, pp. 142-149.

Vilarinho, P., & Simaria, A. (2002). A two-stage Heuristic Method for Balancing Mixed-model Assembly Lines with Parallel Workstations and Zoning Constraints. International Journal of Production Research 40, pp. 1405-1420.

Vilarinho, P., & Simaria, A. (2006). ANTBAL: an Ant Colony Optimization Approach for Balancing Mixed Model Assembly Lines with Parallel Workstations. International Journal of Production Research 44, pp. 291-303.

Wee, T., & Magazine, M. (1981). An Efficient Branch and Bound Algorithm for an Assembly Line Balancing. Waterloo: University of Waterloo.

Xiaobo, Z., & Zhou, Z. (1997). Algorithms for Toyota´s Goal of Sequencing Mixed-models on an Assembly Line with Multiple Workstations. The Journal of the Operational Research Society 50, pp. 704-710.

APPENDIX A

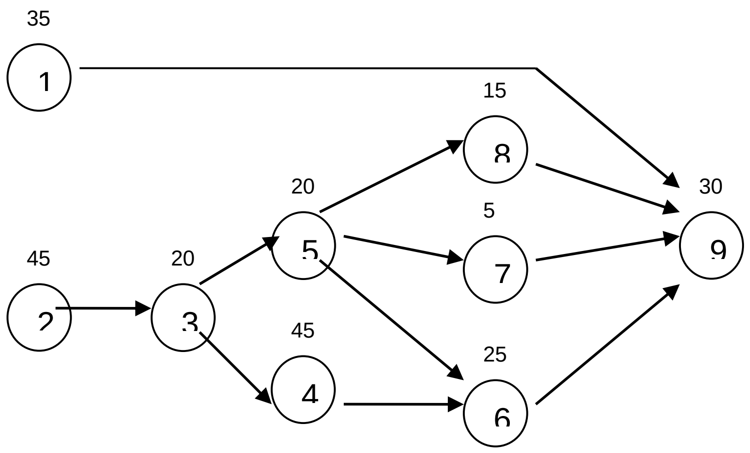


Figure A1: Aase’s Problem Precedence Diagram

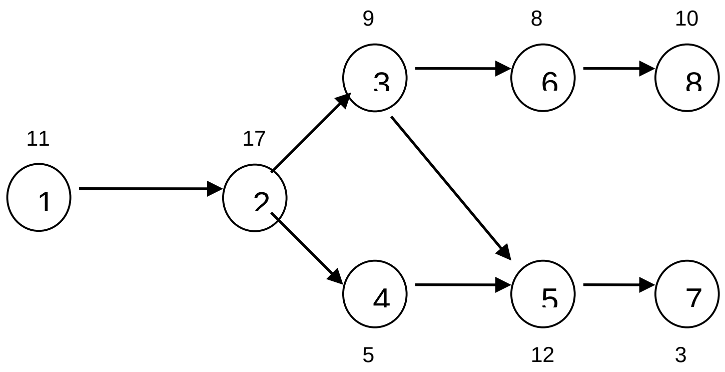


Figure A2: Bowman’s Problem Precedence Diagram

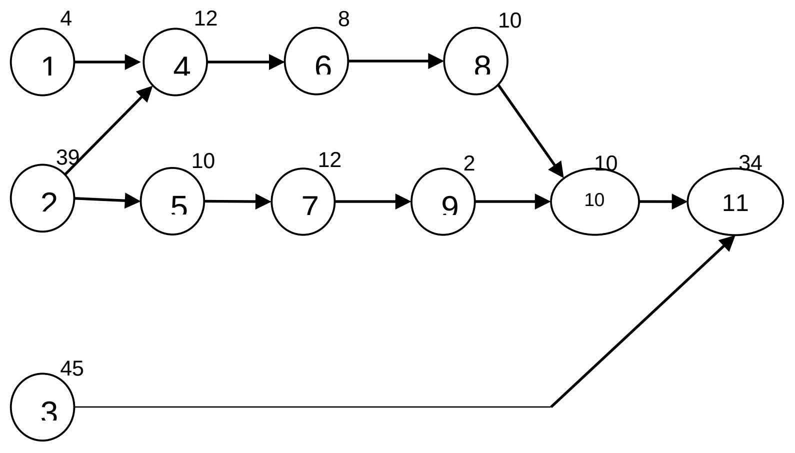


Figure A3: Dar-El's Problem Precedence Diagram

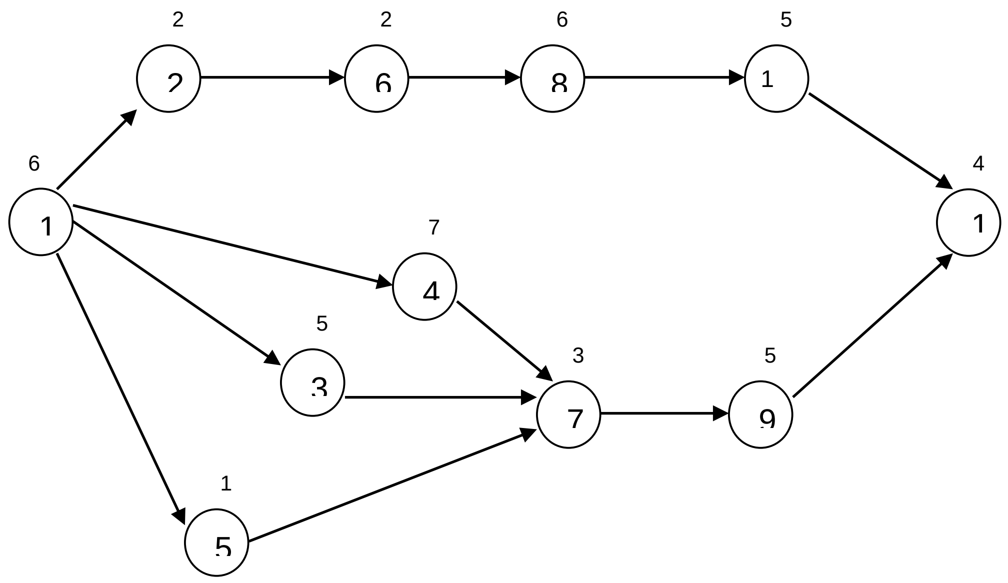
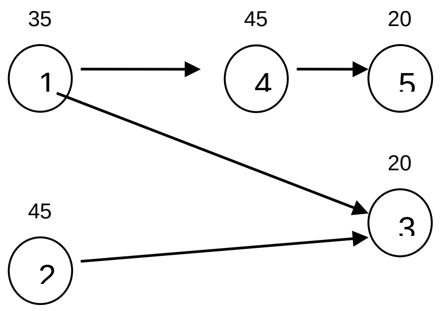


Figure A4: Jackson's Problem Precedence Diagram



-

Figure A5: Johnson’s Problem Precedence Diagram

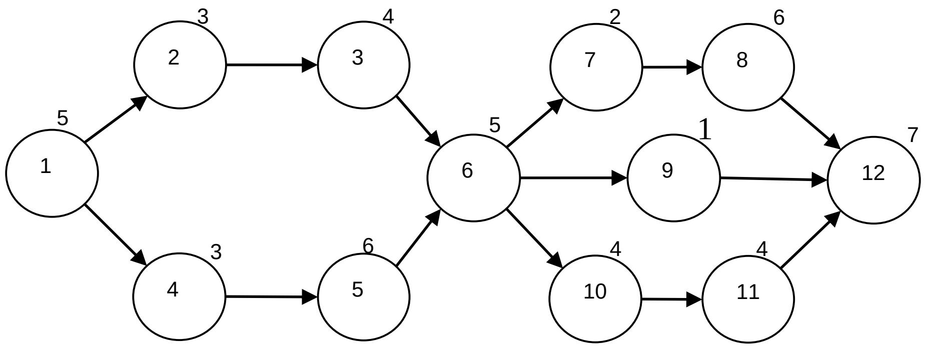


Figure A6: Ponnambalam, Aravindan and Naidu’s Problem Precedence Diagram

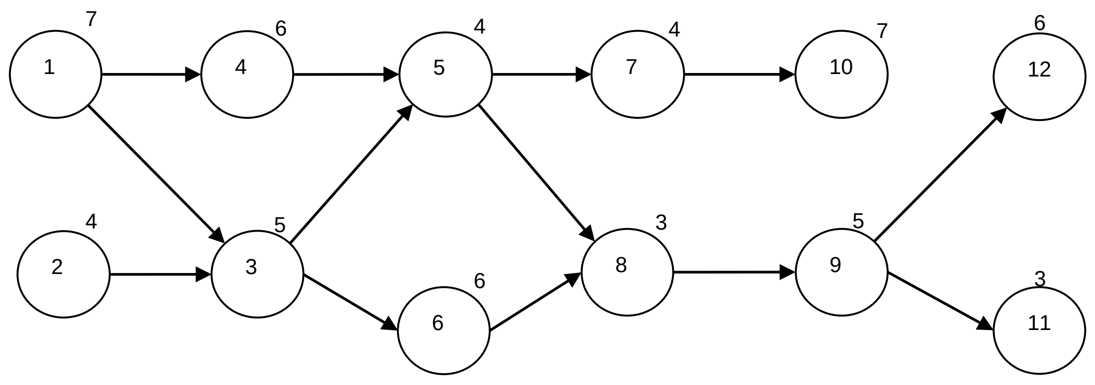


Figure A7: Scholl and Klein’s Problem Precedence Diagram

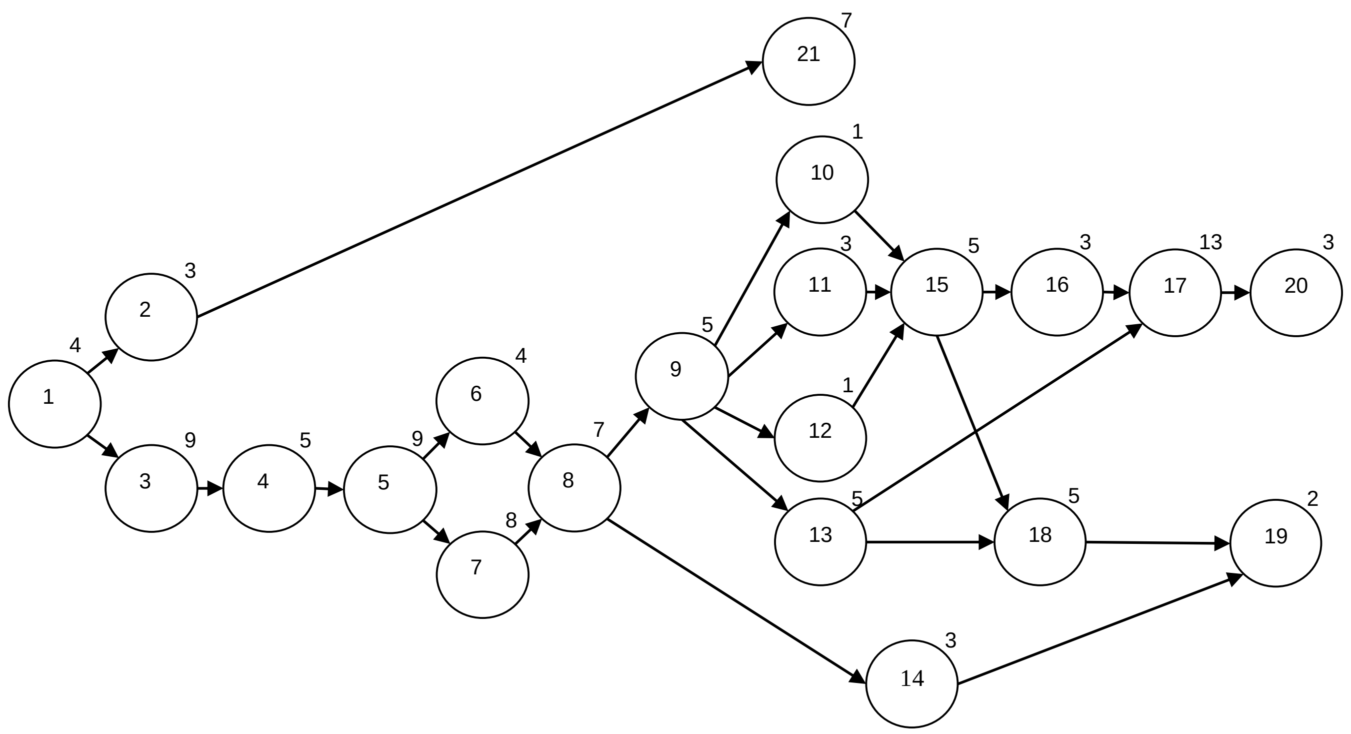


Figure A8: Tonge’s Problem Precedence Diagram

APPENDIX B

Table B1: Enumeration Process Workstation 4 Option 1.1.2

	V	Assign	Tr	V_2
Ws 4 Op 1.1.2	3, 5	3	2	5
	3, 5	5	4	3

Table B2: Enumeration Process Workstation 4 Option 1.1.3

	V	Assign	Tr	V_2
Ws 4 Op 1.1.3	3, 5	3	2	5
	3, 5	5	4	3

Table B3: Enumeration Process Workstation 4 Option 1.1.4

	V	Assign	Tr	V_2
Ws 4 Op 1.1.4	3, 4	3	1	0
	3, 4	4	3	0

Table B4: Enumeration Process Workstation 5 Options 1.1.4.1 and 1.1.4.2

	V	Assign	Tr	V_2
Ws 5 Op 1.1.4.1	4	4	3	0
	V	Assign	Tr	V_2
Ws 5 Op 1.1.4.2	3	3	1	0

Table B5: Enumeration Process Workstation 3 Option 1.2

	V	Assign	Tr	V_2	Tr2
Ws 3 Op 1.2	2, 5, 6	2	0	0	0
	2, 5, 6	5	4	6	1
	2, 5, 6	6	3	4	0
				5	1

Table B6: Enumeration Process Workstation 4 Option 1.2.1

	V	Assign	Tr	V_2	Tr2
Ws 4 Op 1.2.1	4, 5, 6	4	3	5	1
				6	0
	4, 5, 6	5	4	4	1
				6	1
	4, 5, 6	6	3	4	0
				5	1

Table B7: Enumeration Process Workstation 5 Options 1.2.1.1 to 1.2.1.6

	V	Assign	Tr	V_2
Ws 5 Op 1.2.1.1	5	5	4	0
	V	Assign	Tr	V_2
Ws 5 Op 1.2.1.2	6	6	3	0
	V	Assign	Tr	V_2
Ws 5 Op 1.2.1.3	6	6	3	0
	V	Assign	Tr	V_2
Ws 5 Op 1.2.1.4	4	4	3	0
	V	Assign	Tr	V_2
Ws 5 Op 1.2.1.5	3	3	1	0
	V	Assign	Tr	V_2
Ws 5 Op 1.2.1.6	4	4	3	0

Table B8: Enumeration Process Workstation 4 Option 1.2.2

	V	Assign	Tr	V_2
Ws 4 Op 1.2.2	2, 4	2	0	0
	2, 4	4	3	0

Table B9: Enumeration Process Workstation 5 Option 1.2.2.1

	V	Assign	Tr	V_2
Ws 5 Op 1.2.2.1	4	4	3	0
	V	Assign	Tr	V_2
Ws 5 Op 1.2.2.1	2	2	1	0

Table B10: Enumeration Process Workstation 4 Option 1.2.3

	V	Assign	Tr	V_2
Ws 4 Op 1.2.3	2, 5	2	0	0
	2, 5	5	4	0

Table B11: Enumeration Process Workstation 5 Options 1.2.3.1 and 1.2.3.2

	V	Assign	Tr	V_2
Ws 5 Op 1.2.3.1	5	5	4	0
	V	Assign	Tr	V_2
Ws 5 Op 1.2.3.2	2	2	0	0

Table B12: Enumeration Process Workstation 4 Option 1.2.4

	V	Assign	Tr	V_2
Ws 4 Op 1.2.4	2, 4	2	0	0
	2, 4	4	3	0

Table B13: Enumeration Process Workstation 5 Options 1.2.4.1 and 1.2.4.2

	V	Assign	Tr	V_2
Ws 5 Op 1.2.4.1	4	4	3	0
	V	Assign	Tr	V_2
Ws 5 Op 1.2.4.2	2	2	0	0

Table B14: Enumeration Process Workstation 3 Option 1.3

	V	Assign	Tr	V_2
Ws 3 Op 1.3	2, 3, 5	2	0	0
	2, 3, 5	3	1	0
	2, 3, 5	5	4	0

Table B15: Enumeration Process Workstation 4 Option 1.3.1

	V	Assign	Tr	V_2
Ws 4 Op 1.3.1	3, 5	3	1	0
	3, 5	5	4	0

Table B16: Enumeration Process Workstation 5 Options 1.3.1.1 and 1.3.1.2

	V	Assign	Tr	V_2
Ws 5 Op 1.3.1.1	5	5	4	0
Ws 5 Op 1.3.1.2	V	Assign	Tr	V_2
	3	3	1	0

Table B17: Enumeration Process Workstation 4 Option 1.3.2

	V	Assign	Tr	V_2
Ws 4 Op 1.3.2	2, 5	2	0	0
	2, 5	5	4	0

Table B18: Enumeration Process Workstation 5 and 5 Options 1.3.2.1 and 1.3.2

	V	Assign	Tr	V_2
Ws 5 Op 1.3.2.1	5	5	4	0
Ws 4 Op 1.3.2	V	Assign	Tr	V_2
	2	2	0	0

Table B19: Enumeration Process Workstation 4 Option 1.3.3

	V	Assign	Tr	V_2
Ws 4 Op 1.3.3	2, 3	2	0	0
	2, 3	3	1	0

Table B20: Enumeration Process Workstations 5 and 4 Options 1.3.3.1 and 1.3.3

	V	Assign	Tr	V_2
Ws 5 Op 1.3.3.1	3	3	1	0
	V	Assign	Tr	V_2
Ws 4 Op 1.3.3	2	2	0	0

Table B21: Enumeration Process Workstation 3 Option 1.4

	V	Assign	Tr	V_2
Ws 3 Op 1.4	2, 3, 4	2	0	0
	2, 3, 4	3	1	0
	2, 3, 4	4	3	0

Table B22: Enumeration Process Workstation 4 Option 1.4.1

	V	Assign	Tr	V_2
Ws 4 Op 1.4.1	3, 4	3	1	0
	3, 4	4	3	0

Table B23: Enumeration Process Workstation 5 Options 1.4.1.1 and 1.4.1.2

	V	Assign	Tr	V_2
Ws 5 Op 1.4.1.1	4	4	3	0
	V	Assign	Tr	V_2
Ws 5 Op 1.4.1.2	3, 4	3	1	0

Table B24: Enumeration Process Workstation 4 Option 1.4.2

	V	Assign	Tr	V_2
Ws 4 Op 1.4.2	2, 4	2	0	0
	2, 4	4	3	0

Table B25: Enumeration Process Workstation 5 Options 1.4.2.1 and 1.4.2.2

	V	Assign	Tr	V_2
Ws 5 Op 1.4.2.1	4	4	3	0
	V	Assign	Tr	V_2
Ws 5 Op 1.4.2.2	2, 4	2	0	0

Table B26: Enumeration Process Workstation 4 Option 1.4.3

	V	Assign	Tr	V_2
Ws 4 Op 1.4.3	2, 3	2	0	0
	2, 3	3	1	0

Table B27: Enumeration Process Workstation 5 Options 1.4.3.1 and 1.4.3.2

	V	Assign	Tr	V_2
Ws 5 Op 1.4.3.1	2, 3	3	1	0
	V	Assign	Tr	V_2
Ws 5 Op 1.4.3.2	2, 3	2	0	0

Table B28: Enumeration Process Workstation 2 Option 2

	V	Assign	Tr	V_2
Ws 2 Op 2	1, 2, 5	1	2	5
	1, 2, 5	2	0	0
	1, 2, 5	5	1	0

Table B29: Enumeration Process Workstation 3 Option 2.1

	V	Assign	Tr	V_2
Ws 3 Op 2.1	2, 3	2	0	0
	2, 3	3	1	0

Table B30: Enumeration Process Workstation 4 Options 2.1.1 and 2.1.2

	V	Assign	Tr	V_2
Ws 4 Op 2.1.1	3	3	1	0
Ws 4 Op 2.1.2	V	Assign	Tr	V_2
	2, 3	2	0	0

Table B31: Enumeration Process Workstation 3 Option 2.2

	V	Assign	Tr	V_2
Ws 3 Op 2.2	1, 5	1	2	5
	1, 5	5	4	1

Table B32: Enumeration Process Workstation 4 Options 2.2.1 and 2.2.2

	V	Assign	Tr	V_2
Ws 4 Op 2.2.1	3	3	1	0
Ws 4 Op 2.2.2	V	Assign	Tr	V_2
	3	3	1	0

Table B33: Enumeration Process Workstation 3 Option 2.3

	V	Assign	Tr	V_2
Ws 3 Op 2.3	1, 2, 3	1	2	0
	1, 2, 3	2	0	0
	1, 2, 3	3	1	0

Table B34: Enumeration Process Workstation 4 Option 2.3.1

Ws 4 Op 2.3.1	V	Assign	Tr	V_2
	2, 3	2	0	0
	2, 3	3	1	0

Table B35: Enumeration Process Workstation 5 Options 2.3.1.1 and 2.3.1.2

Ws 5 Op 2.3.1.1	V	Assign	Tr	V_2
	3	3	1	0
Ws 5 Op 2.3.1.2	V	Assign	Tr	V_2
	2	2	0	0

Table B36: Enumeration Process Workstation 4 Option 2.3.2

Ws 4 Op 2.3.2	V	Assign	Tr	V_2
	1, 3	1	2	0
	1, 3	3	1	0

Table B37: Enumeration Process Workstation 5 Options 2.3.2.1 and 2.3.2.2

Ws 5 Op 2.3.2.1	V	Assign	Tr	V_2
	3	3	1	0
Ws 5 Op 2.3.2.2	V	Assign	Tr	V_2
	1	1	2	0

Table B38: Enumeration Process Workstation 4 Option 2.3.3

Ws 4 Op 2.3.3	V	Assign	Tr	V_2
	1, 2	1	2	0
	1, 2	2	0	0

Table B39: Enumeration Process Workstation 5 Options 2.3.3.1 and 2.3.3.2

Ws 5 Op 2.3.3.1	V	Assign	Tr	V_2
	2	2	0	0
Ws 5 Op 2.3.3.2	V	Assign	Tr	V_2
	1	1	2	0

Table B40: Enumeration Process Workstation 2 Option 3

	V	Assign	Tr	V_2
Ws 2 Op 3	1, 3, 4	1	2	0
	1, 3, 4	3	1	0
	1, 3, 4	4	1	0

Table B41: Enumeration Process Workstation 3 Option 3.1

	V	Assign	Tr	V_2
Ws 3 Op 3.1	2, 3, 4	2	0	0
	2, 3, 4	3	1	0
	2, 3, 4	4	3	0

Table B42: Enumeration Process Workstation 4 Option 3.1.1

	V	Assign	Tr	V_2
Ws 4 Op 3.1.1	3, 4	3	1	0
	3, 4	4	3	0

Table B43: Enumeration Process Workstation 5 Options 3.1.1.1 and 3.1.1.2

	V	Assign	Tr	V_2
Ws 5 Op 3.1.1.1	4	4	3	0
Ws 5 Op 3.1.1.2	V	Assign	Tr	V_2
	3	3	1	0

Table B44: Enumeration Process Workstation 4 Option 3.1.2

	V	Assign	Tr	V_2
Ws 4 Op 3.1.2	2, 4	2	0	0
	2, 4	4	3	0

Table B45: Enumeration Process Workstation 5 Options 3.1.2.1 and 3.1.2.2

	V	Assign	Tr	V_2
Ws 5 Op 3.1.2.1	4	4	3	0
Ws 5 Op 3.1.2.2	V	Assign	Tr	V_2
	2	2	0	0

Table B46: Enumeration Process Workstation 4 Option 3.1.3

	V	Assign	Tr	V_2
Ws 4 Op 3.1.3	2, 3	2	0	0
	2, 3	3	1	0

Table B47: Enumeration Process Workstation 5 Options 3.1.3.1 and 3.1.3.2

	V	Assign	Tr	V_2
Ws 5 Op 3.1.3.1	3	3	1	0
Ws 5 Op 3.1.3.2	V	Assign	Tr	V_2
	2	2	0	0

Table B48: Enumeration Process Workstation 3 Option 3.2

	V	Assign	Tr	V_2
Ws 3 Op 3.2	1, 4	1	2	0
	1, 4	4	3	0

Table B49: Enumeration Process Workstation 4 Option 3.2.1

	V	Assign	Tr	V_2
Ws 4 Op 3.2.1	2, 4	2	0	0
	2, 4	4	3	0

Table B50: Enumeration Process Workstations 5 and 4 Options 3.2.1.1 and 3.2.1.2

Ws 5	V	Assign	Tr	V_2
Op 3.2.1.1	4	4	3	0
Ws 4	V	Assign	Tr	V_2
Op 3.2.1.2	2	2	0	0

Table B51: Enumeration Process Workstation 4 Option 3.2.2

	V	Assign	Tr	V_2
Ws 4 Op 3.2.2	1, 2	1	2	0
	1, 2	2	0	0

Table B52 Enumeration Process Workstation 5 Options 3.2.2.1 and 3.2.2.2

Ws 5	V	Assign	Tr	V_2
Op 3.2.2.1	2	2	0	0
Ws 5	V	Assign	Tr	V_2
Op 3.2.2.2	1	1	2	0

Table B53: Enumeration Process Workstation 3 Option 3.3

	V	Assign	Tr	V_2
Ws 3 Op 3.3	1, 2, 3	1	2	0
	1, 2, 3	2	0	0
	1, 2, 3	3	1	0

Table B54: Enumeration Process Workstation 4 Option 3.3.1

	V	Assign	Tr	V_2
Ws 4 Op 3.3.1	2, 3	2	0	0
	2, 3	3	1	0

Table B55: Enumeration Process Workstation 5 Options 3.3.1.1 and 3.3.1.2

Ws 5 Op 3.3.1.1	V	Assign	Tr	V_2
	3	3	1	0
Ws 5 Op 3.3.1.2	V	Assign	Tr	V_2
	2	2	0	0

Table B56: Enumeration Process Workstation 4 Option 3.3.2

Ws 4 Op 3.3.2	V	Assign	Tr	V_2
	1, 3	1	2	0
	1, 3	3	1	0

Table B57: Enumeration Process Workstations 5 and 4 Options 3.3.2.1 to 3.3.3

Ws 5 Op 3.3.2.1	V	Assign	Tr	V_2
	3	3	1	0
Ws 5 Op 3.3.2.2	V	Assign	Tr	V_2
	1	1	2	0
Ws 4 Op 3.3.3	V	Assign	Tr	V_2
	1, 2	1	2	0
	1, 2	2	0	0

Table B58: Enumeration Process Workstation 5 Options 3.3.3.1 and 3.3.3.2

Ws 5 Op 3.3.3.1	V	Assign	Tr	V_2
	2	2	0	0
Ws 5 Op 3.3.3.2	V	Assign	Tr	V_2
	1	1	2	0

APPENDIX C

Integer Programming Solution for the LBP Using the Software LINDO

MIN
X_a1+2X_a2+3X_a3+4X_a4+5X_a5+X_b1+2X_b2+3X_b3+4X_b4+5X_b5+X_c1+2X_c2+3X_c3+4X_c4+5X_c5+X_d1+2X_d2+3X_d3+4X_d4+5X_d5+X_e1+2X_e2+3X_e3+4X_e4+5X_e5

SUBJECT TO

Applying the cycle time restriction we get:

40X_a1+75X_b1+50X_c1+35X_d1+80X_e1<=100

40X_a2+75X_b2+50X_c2+35X_d2+80X_e2<=100

40X_a3+75X_b3+50X_c3+35X_d3+80X_e3<=100

40X_a4+75X_b4+50X_c4+35X_d4+80X_e4<=100

40X_a5+75X_b5+50X_c5+35X_d5+80X_e5<=100

And after applying unit assignment restrictions:

X_a1+X_a2+X_a3+X_a4+X_a5=1

X_b1+X_b2+X_b3+X_b4+X_b5=1

X_c1+X_c2+X_c3+X_c4+X_c5=1

X_d1+X_d2+X_d3+X_d4+X_d5=1

X_e1+X_e2+X_e3+X_e4+X_e5=1

And finally, the precedence restrictions:

X_b1-X_a1<=0

X_b2-X_a1-X_a2<=0

X_b3-X_a1-X_a2-X_a3<=0

X_b4-X_a1-X_a2-X_a3-X_a4<=0

X_b5-X_a1-X_a2-X_a3-X_a4-X_a5<=0

Since *a* is the immediate predecessor of *c*, then the following complies:

X_c1-X_a1<=0

X_c2-X_a1-X_a2<=0

X_c3-X_a1-X_a2-X_a3<=0

X_c4-X_a1-X_a2-X_a3-X_a4<=0

X_c5-X_a1-X_a2-X_a3-X_a4-X_a5<=0

Likewise, *c* is the predecessor of *d*, so:

X_d1-X_c1<=0

X_d2-X_c1-X_c2<=0

X_d3-X_c1-X_c2-X_c3<=0

X_d4-X_c1-X_c2-X_c3-X_c4<=0

X_d5-X_c1-X_c2-X_c3-X_c4-X_c5<=0

The final restriction says that *e* has two predecessors, *b* and *d*. So, applying these restrictions results in:

X_e1-X_b1-X_d1<=0

X_e2-X_b1-X_d1-X_b2-X_d2<=0

X_e3-X_b1-X_d1-X_b2-X_d2-X_b3-X_d3<=0

X_e4-X_b1-X_d1-X_b2-X_d2-X_b3-X_d3-X_b4-X_d4<=0

X_e5-X_b1-X_d1-X_b2-X_d2-X_b3-X_d3-X_b4-X_d4-X_b5-X_d5<=0

```
END
 INT 25

LP OPTIMUM FOUND AT STEP     12
 OBJECTIVE VALUE =   8.58823490

 FIX ALL VARS.(   10)  WITH RC >   1.00000
 SET     X_C1 TO >=     1 AT    1, BND=  -10.16     TWIN= -10.31         57
 SET     X_B3 TO <=     0 AT    2, BND=  -10.19     TWIN= -10.50         62
 SET     X_E3 TO <=     0 AT    3, BND=  -11.00     TWIN= -10.43         67

 NEW INTEGER SOLUTION OF    11.0000000     AT BRANCH      3 PIVOT      67
 BOUND ON OPTIMUM:  10.31250
 DELETE     X_E3 AT LEVEL     3
 DELETE     X_B3 AT LEVEL     2
 DELETE     X_C1 AT LEVEL     1
 ENUMERATION COMPLETE. BRANCHES=     3 PIVOTS=      67

 LAST INTEGER SOLUTION IS THE BEST FOUND
 RE-INSTALLING BEST SOLUTION...

       OBJECTIVE FUNCTION VALUE

       1)      11.00000

  VARIABLE        VALUE          REDUCED COST
      X_A1         1.000000          1.000000
      X_A2         0.000000          2.000000
```

```
   X_A3        0.000000         3.000000
   X_A4        0.000000         4.000000
   X_A5        0.000000         5.000000
   X_B1        0.000000         1.000000
   X_B2        1.000000         2.000000
   X_B3        0.000000         3.000000
   X_B4        0.000000         4.000000
   X_B5        0.000000         5.000000
   X_C1        1.000000         1.000000
   X_C2        0.000000         2.000000
   X_C3        0.000000         3.000000
   X_C4        0.000000         4.000000
   X_C5        0.000000         5.000000
   X_D1        0.000000         1.000000
   X_D2        0.000000         2.000000
   X_D3        1.000000         3.000000
   X_D4        0.000000         4.000000
   X_D5        0.000000         5.000000
   X_E1        0.000000         1.000000
   X_E2        0.000000         2.000000
   X_E3        0.000000         3.000000
   X_E4        1.000000         4.000000
   X_E5        0.000000         5.000000


 ROW   SLACK OR SURPLUS     DUAL PRICES
     2)      10.000000         0.000000
   3)       25.000000          0.000000
   4)       65.000000          0.000000
   5)       20.000000          0.000000
   6)      100.000000          0.000000
   7)        0.000000          0.000000
   8)        0.000000          0.000000
   9)        0.000000          0.000000
  10)        0.000000          0.000000
  11)        0.000000          0.000000
  12)        1.000000          0.000000
  13)        0.000000          0.000000
  14)        1.000000          0.000000
  15)        1.000000          0.000000
  16)        1.000000          0.000000
  17)        0.000000          0.000000
  18)        1.000000          0.000000
  19)        1.000000          0.000000
  20)        1.000000          0.000000
  21)        1.000000          0.000000
  22)        1.000000          0.000000
  23)        1.000000          0.000000
  24)        0.000000          0.000000
  25)        1.000000          0.000000
  26)        1.000000          0.000000
  27)        0.000000          0.000000
  28)        1.000000          0.000000
  29)        2.000000          0.000000
  30)        1.000000          0.000000
```

```
     31)        2.000000          0.000000


NO. ITERATIONS=      69

BRANCHES=    3 DETERM.=  1.000E    0
```

APPENDIX D

Table D1: Coded Values for Chromosomes 10-14

10	-1	11	-0.97778	12	-0.95556	13	-0.93333	14	-0.91111
Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation
-1.00	-1.00								
0.00	0.00	-0.18	-0.18	-0.33	-0.33	-0.46	-0.46	-0.57	-0.57
1.00	1.00	0.73	0.73	0.50	0.50	0.31	0.31	0.14	0.14
								0.86	0.86

Table D2: Coded Values for Chromosomes 15-19

15	-0.88889	16	-0.86667	17	-0.84444	18	-0.82222	19	-0.8
Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation
-0.67	-0.67	-0.75	-0.75	-0.82	-0.82	-0.89	-0.89	-0.95	-0.95
0.00	0.00	-0.13	-0.13	-0.24	-0.24	-0.33	-0.33	-0.42	-0.42
0.67	0.67	0.50	0.50	0.35	0.35	0.22	0.22	0.11	0.11
				0.94	0.94	0.78	0.78	0.63	0.63

Table D3: Coded Values for Chromosomes 20-24

20	-0.77778	21	-0.75556	22	-0.73333	23	-0.71111	24	-0.68889
Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation
-1.00	-1.00								
-0.50	-0.50	-0.57	-0.57	-0.64	-0.64	-0.70	-0.70	-0.75	-0.75
0.00	0.00	-0.10	-0.10	-0.18	-0.18	-0.26	-0.26	-0.33	-0.33
0.50	0.50	0.38	0.38	0.27	0.27	0.17	0.17	0.08	0.08
1.00	1.00	0.86	0.86	0.73	0.73	0.61	0.61	0.50	0.50
								0.92	0.92

Table D4: Coded Values for Chromosomes 25-29

25	-0.66667	26	-0.64444	27	-0.62222	28	-0.6	29	-0.57778
Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation
-0.80	-0.80	-0.85	-0.85	-0.89	-0.89	-0.93	-0.93	-0.97	-0.97
-0.40	-0.40	-0.46	-0.46	-0.52	-0.52	-0.57	-0.57	-0.62	-0.62
0.00	0.00	-0.08	-0.08	-0.15	-0.15	-0.21	-0.21	-0.28	-0.28
0.40	0.40	0.31	0.31	0.22	0.22	0.14	0.14	0.07	0.07
0.80	0.80	0.69	0.69	0.59	0.59	0.50	0.50	0.41	0.41
				0.96	0.96	0.86	0.86	0.76	0.76

Table D5: Coded Values for Chromosomes 30-34

30	-0.55556	31	-0.53333	32	-0.51111	33	-0.48889	34	-0.46667
Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation
-1.00	-1.00								
-0.67	-0.67	-0.71	-0.71	-0.75	-0.75	-0.79	-0.79	-0.82	-0.82
-0.33	-0.33	-0.39	-0.39	-0.44	-0.44	-0.48	-0.48	-0.53	-0.53
0.00	0.00	-0.06	-0.06	-0.13	-0.13	-0.18	-0.18	-0.24	-0.24
0.33	0.33	0.26	0.26	0.19	0.19	0.12	0.12	0.06	0.06
0.67	0.67	0.58	0.58	0.50	0.50	0.42	0.42	0.35	0.35
1.00	1.00	0.90	0.90	0.81	0.81	0.73	0.73	0.65	0.65
								0.94	0.94

Table D6: Coded Values for Chromosomes 35-39

35	-0.44444	36	-0.42222	37	-0.4	38	-0.37778	39	-0.35556
Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation
-0.86	-0.86	-0.89	-0.89	-0.92	-0.92	-0.95	-0.95	-0.97	-0.97
-0.57	-0.57	-0.61	-0.61	-0.65	-0.65	-0.68	-0.68	-0.72	-0.72
-0.29	-0.29	-0.33	-0.33	-0.38	-0.38	-0.42	-0.42	-0.46	-0.46
0.00	0.00	-0.06	-0.06	-0.11	-0.11	-0.16	-0.16	-0.21	-0.21
0.29	0.29	0.22	0.22	0.16	0.16	0.11	0.11	0.05	0.05
0.57	0.57	0.50	0.50	0.43	0.43	0.37	0.37	0.31	0.31
0.86	0.86	0.78	0.78	0.70	0.70	0.63	0.63	0.56	0.56
				0.97	0.97	0.89	0.89	0.82	0.82

Table D7: Coded Values for Chromosomes 40-44

40	-0.33333	41	-0.31111	42	-0.28889	43	-0.26667	44	-0.24444
Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation
-1.00	-1.00								
-0.75	-0.75	-0.78	-0.78	-0.81	-0.81	-0.84	-0.84	-0.86	-0.86
-0.50	-0.50	-0.54	-0.54	-0.57	-0.57	-0.60	-0.60	-0.64	-0.64
-0.25	-0.25	-0.29	-0.29	-0.33	-0.33	-0.37	-0.37	-0.41	-0.41
0.00	0.00	-0.05	-0.05	-0.10	-0.10	-0.14	-0.14	-0.18	-0.18
0.25	0.25	0.20	0.20	0.14	0.14	0.09	0.09	0.05	0.05
0.50	0.50	0.44	0.44	0.38	0.38	0.33	0.33	0.27	0.27
0.75	0.75	0.68	0.68	0.62	0.62	0.56	0.56	0.50	0.50
1.00	1.00	0.93	0.93	0.86	0.86	0.79	0.79	0.73	0.73
								0.95	0.95

Table D8: Coded Values for Chromosomes 45-49

45	-0.22222	46	-0.2	47	-0.17778	48	-0.15556	49	-0.13333
Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation
-0.89	-0.89	-0.91	-0.91	-0.94	-0.94	-0.96	-0.96	-0.98	-0.98
-0.67	-0.67	-0.70	-0.70	-0.72	-0.72	-0.75	-0.75	-0.78	-0.78
-0.44	-0.44	-0.48	-0.48	-0.51	-0.51	-0.54	-0.54	-0.57	-0.57
-0.22	-0.22	-0.26	-0.26	-0.30	-0.30	-0.33	-0.33	-0.37	-0.37
0.00	0.00	-0.04	-0.04	-0.09	-0.09	-0.13	-0.13	-0.16	-0.16
0.22	0.22	0.17	0.17	0.13	0.13	0.08	0.08	0.04	0.04
0.44	0.44	0.39	0.39	0.34	0.34	0.29	0.29	0.24	0.24
0.67	0.67	0.61	0.61	0.55	0.55	0.50	0.50	0.45	0.45
0.89	0.89	0.83	0.83	0.77	0.77	0.71	0.71	0.65	0.65
				0.98	0.98	0.92	0.92	0.86	0.86

Table D9: Coded Values for Chromosomes 50-54

50	-0.11111	51	-0.08889	52	-0.06667	53	-0.04444	54	-0.02222
Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation
-1.00	-1.00								
-0.80	-0.80	-0.82	-0.82	-0.85	-0.85	-0.87	-0.87	-0.89	-0.89
-0.60	-0.60	-0.63	-0.63	-0.65	-0.65	-0.68	-0.68	-0.70	-0.70
-0.40	-0.40	-0.43	-0.43	-0.46	-0.46	-0.49	-0.49	-0.52	-0.52
-0.20	-0.20	-0.24	-0.24	-0.27	-0.27	-0.30	-0.30	-0.33	-0.33
0.00	0.00	-0.04	-0.04	-0.08	-0.08	-0.11	-0.11	-0.15	-0.15
0.20	0.20	0.16	0.16	0.12	0.12	0.08	0.08	0.04	0.04
0.40	0.40	0.35	0.35	0.31	0.31	0.26	0.26	0.22	0.22
0.60	0.60	0.55	0.55	0.50	0.50	0.45	0.45	0.41	0.41
0.80	0.80	0.75	0.75	0.69	0.69	0.64	0.64	0.59	0.59
1.00	1.00	0.94	0.94	0.88	0.88	0.83	0.83	0.78	0.78
								0.96	0.96

Table D10: Coded Values for Chromosomes 55-59

55	0	56	0.022222	57	0.044444	58	0.066667	59	0.088889
Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation
-0.91	-0.91	-0.93	-0.93	-0.95	-0.95	-0.97	-0.97	-0.98	-0.98
-0.73	-0.73	-0.75	-0.75	-0.77	-0.77	-0.79	-0.79	-0.81	-0.81
-0.55	-0.55	-0.57	-0.57	-0.60	-0.60	-0.62	-0.62	-0.64	-0.64
-0.36	-0.36	-0.39	-0.39	-0.42	-0.42	-0.45	-0.45	-0.47	-0.47
-0.18	-0.18	-0.21	-0.21	-0.25	-0.25	-0.28	-0.28	-0.31	-0.31
0.00	0.00	-0.04	-0.04	-0.07	-0.07	-0.10	-0.10	-0.14	-0.14
0.18	0.18	0.14	0.14	0.11	0.11	0.07	0.07	0.03	0.03
0.36	0.36	0.32	0.32	0.28	0.28	0.24	0.24	0.20	0.20
0.55	0.55	0.50	0.50	0.46	0.46	0.41	0.41	0.37	0.37
0.73	0.73	0.68	0.68	0.63	0.63	0.59	0.59	0.54	0.54
0.91	0.91	0.86	0.86	0.81	0.81	0.76	0.76	0.71	0.71
				0.98	0.98	0.93	0.93	0.88	0.88

Table D11: Coded Values for Chromosomes 60-64

60	0.111111	61	0.133333	62	0.155556	63	0.177778	64	0.2
Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation
-1.00	-1.00								
-0.83	-0.83	-0.85	-0.85	-0.87	-0.87	-0.89	-0.89	-0.91	-0.91
-0.67	-0.67	-0.69	-0.69	-0.71	-0.71	-0.73	-0.73	-0.75	-0.75
-0.50	-0.50	-0.52	-0.52	-0.55	-0.55	-0.57	-0.57	-0.59	-0.59
-0.33	-0.33	-0.36	-0.36	-0.39	-0.39	-0.41	-0.41	-0.44	-0.44
-0.17	-0.17	-0.20	-0.20	-0.23	-0.23	-0.25	-0.25	-0.28	-0.28
0.00	0.00	-0.03	-0.03	-0.06	-0.06	-0.10	-0.10	-0.13	-0.13
0.17	0.17	0.13	0.13	0.10	0.10	0.06	0.06	0.03	0.03
0.33	0.33	0.30	0.30	0.26	0.26	0.22	0.22	0.19	0.19
0.50	0.50	0.46	0.46	0.42	0.42	0.38	0.38	0.34	0.34
0.67	0.67	0.62	0.62	0.58	0.58	0.54	0.54	0.50	0.50
0.83	0.83	0.79	0.79	0.74	0.74	0.70	0.70	0.66	0.66
1.00	1.00	0.95	0.95	0.90	0.90	0.86	0.86	0.81	0.81
								0.97	0.97

Table D12: Coded Values for Chromosomes 65-69

65	0.222222	66	0.244444	67	0.266667	68	0.288889	69	0.311111
Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation
-0.92	-0.92	-0.94	-0.94	-0.96	-0.96	-0.97	-0.97	-0.99	-0.99
-0.77	-0.77	-0.79	-0.79	-0.81	-0.81	-0.82	-0.82	-0.84	-0.84
-0.62	-0.62	-0.64	-0.64	-0.66	-0.66	-0.68	-0.68	-0.70	-0.70
-0.46	-0.46	-0.48	-0.48	-0.51	-0.51	-0.53	-0.53	-0.55	-0.55
-0.31	-0.31	-0.33	-0.33	-0.36	-0.36	-0.38	-0.38	-0.41	-0.41
-0.15	-0.15	-0.18	-0.18	-0.21	-0.21	-0.24	-0.24	-0.26	-0.26
0.00	0.00	-0.03	-0.03	-0.06	-0.06	-0.09	-0.09	-0.12	-0.12
0.15	0.15	0.12	0.12	0.09	0.09	0.06	0.06	0.03	0.03
0.31	0.31	0.27	0.27	0.24	0.24	0.21	0.21	0.17	0.17
0.46	0.46	0.42	0.42	0.39	0.39	0.35	0.35	0.32	0.32
0.62	0.62	0.58	0.58	0.54	0.54	0.50	0.50	0.46	0.46
0.77	0.77	0.73	0.73	0.69	0.69	0.65	0.65	0.61	0.61
0.92	0.92	0.88	0.88	0.84	0.84	0.79	0.79	0.75	0.75
				0.99	0.99	0.94	0.94	0.90	0.90

Table D13: Coded Values for Chromosomes 70-74

70	0.333333	71	0.355556	72	0.377778	73	0.4	74	0.422222
Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation
-1.00	-1.00								
-0.86	-0.86	-0.87	-0.87	-0.89	-0.89	-0.90	-0.90	-0.92	-0.92
-0.71	-0.71	-0.73	-0.73	-0.75	-0.75	-0.77	-0.77	-0.78	-0.78
-0.57	-0.57	-0.59	-0.59	-0.61	-0.61	-0.63	-0.63	-0.65	-0.65
-0.43	-0.43	-0.45	-0.45	-0.47	-0.47	-0.49	-0.49	-0.51	-0.51
-0.29	-0.29	-0.31	-0.31	-0.33	-0.33	-0.36	-0.36	-0.38	-0.38
-0.14	-0.14	-0.17	-0.17	-0.19	-0.19	-0.22	-0.22	-0.24	-0.24
0.00	0.00	-0.03	-0.03	-0.06	-0.06	-0.08	-0.08	-0.11	-0.11
0.14	0.14	0.11	0.11	0.08	0.08	0.05	0.05	0.03	0.03
0.29	0.29	0.25	0.25	0.22	0.22	0.19	0.19	0.16	0.16
0.43	0.43	0.39	0.39	0.36	0.36	0.33	0.33	0.30	0.30
0.57	0.57	0.54	0.54	0.50	0.50	0.47	0.47	0.43	0.43
0.71	0.71	0.68	0.68	0.64	0.64	0.60	0.60	0.57	0.57
0.86	0.86	0.82	0.82	0.78	0.78	0.74	0.74	0.70	0.70
1.00	1.00	0.96	0.96	0.92	0.92	0.88	0.88	0.84	0.84
								0.97	0.97

Table D14: Coded Values for Chromosomes 75-79

75	0.444444	76	0.466667	77	0.488889	78	0.511111	79	0.533333
Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation
-0.93	-0.93	-0.95	-0.95	-0.96	-0.96	-0.97	-0.97	-0.99	-0.99
-0.80	-0.80	-0.82	-0.82	-0.83	-0.83	-0.85	-0.85	-0.86	-0.86
-0.67	-0.67	-0.68	-0.68	-0.70	-0.70	-0.72	-0.72	-0.73	-0.73
-0.53	-0.53	-0.55	-0.55	-0.57	-0.57	-0.59	-0.59	-0.61	-0.61
-0.40	-0.40	-0.42	-0.42	-0.44	-0.44	-0.46	-0.46	-0.48	-0.48
-0.27	-0.27	-0.29	-0.29	-0.31	-0.31	-0.33	-0.33	-0.35	-0.35
-0.13	-0.13	-0.16	-0.16	-0.18	-0.18	-0.21	-0.21	-0.23	-0.23
0.00	0.00	-0.03	-0.03	-0.05	-0.05	-0.08	-0.08	-0.10	-0.10
0.13	0.13	0.11	0.11	0.08	0.08	0.05	0.05	0.03	0.03
0.27	0.27	0.24	0.24	0.21	0.21	0.18	0.18	0.15	0.15
0.40	0.40	0.37	0.37	0.34	0.34	0.31	0.31	0.28	0.28
0.53	0.53	0.50	0.50	0.47	0.47	0.44	0.44	0.41	0.41
0.67	0.67	0.63	0.63	0.60	0.60	0.56	0.56	0.53	0.53
0.80	0.80	0.76	0.76	0.73	0.73	0.69	0.69	0.66	0.66
0.93	0.93	0.89	0.89	0.86	0.86	0.82	0.82	0.78	0.78
				0.99	0.99	0.95	0.95	0.91	0.91

Table D15: Coded Values for Chromosomes 80-84

80	0.555556	81	0.577778	82	0.6	83	0.622222	84	0.644444
Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation
-1.00	-1.00								
-0.88	-0.88	-0.89	-0.89	-0.90	-0.90	-0.92	-0.92	-0.93	-0.93
-0.75	-0.75	-0.77	-0.77	-0.78	-0.78	-0.80	-0.80	-0.81	-0.81
-0.63	-0.63	-0.64	-0.64	-0.66	-0.66	-0.67	-0.67	-0.69	-0.69
-0.50	-0.50	-0.52	-0.52	-0.54	-0.54	-0.55	-0.55	-0.57	-0.57
-0.38	-0.38	-0.40	-0.40	-0.41	-0.41	-0.43	-0.43	-0.45	-0.45
-0.25	-0.25	-0.27	-0.27	-0.29	-0.29	-0.31	-0.31	-0.33	-0.33
-0.13	-0.13	-0.15	-0.15	-0.17	-0.17	-0.19	-0.19	-0.21	-0.21
0.00	0.00	-0.02	-0.02	-0.05	-0.05	-0.07	-0.07	-0.10	-0.10
0.13	0.13	0.10	0.10	0.07	0.07	0.05	0.05	0.02	0.02
0.25	0.25	0.22	0.22	0.20	0.20	0.17	0.17	0.14	0.14
0.38	0.38	0.35	0.35	0.32	0.32	0.29	0.29	0.26	0.26
0.50	0.50	0.47	0.47	0.44	0.44	0.41	0.41	0.38	0.38
0.63	0.63	0.59	0.59	0.56	0.56	0.53	0.53	0.50	0.50
0.75	0.75	0.72	0.72	0.68	0.68	0.65	0.65	0.62	0.62
0.88	0.88	0.84	0.84	0.80	0.80	0.77	0.77	0.74	0.74
1.00	1.00	0.96	0.96	0.93	0.93	0.89	0.89	0.86	0.86
								0.98	0.98

Table D16: Coded Values for Chromosomes 85-89

85	0.666667	86	0.688889	87	0.711111	88	0.733333	89	0.755556
Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation
-0.94	-0.94	-0.95	-0.95	-0.97	-0.97	-0.98	-0.98	-0.99	-0.99
-0.82	-0.82	-0.84	-0.84	-0.85	-0.85	-0.86	-0.86	-0.88	-0.88
-0.71	-0.71	-0.72	-0.72	-0.74	-0.74	-0.75	-0.75	-0.76	-0.76
-0.59	-0.59	-0.60	-0.60	-0.62	-0.62	-0.64	-0.64	-0.65	-0.65
-0.47	-0.47	-0.49	-0.49	-0.51	-0.51	-0.52	-0.52	-0.54	-0.54
-0.35	-0.35	-0.37	-0.37	-0.39	-0.39	-0.41	-0.41	-0.43	-0.43
-0.24	-0.24	-0.26	-0.26	-0.28	-0.28	-0.30	-0.30	-0.31	-0.31
-0.12	-0.12	-0.14	-0.14	-0.16	-0.16	-0.18	-0.18	-0.20	-0.20
0.00	0.00	-0.02	-0.02	-0.05	-0.05	-0.07	-0.07	-0.09	-0.09
0.12	0.12	0.09	0.09	0.07	0.07	0.05	0.05	0.02	0.02
0.24	0.24	0.21	0.21	0.18	0.18	0.16	0.16	0.13	0.13
0.35	0.35	0.33	0.33	0.30	0.30	0.27	0.27	0.25	0.25
0.47	0.47	0.44	0.44	0.41	0.41	0.39	0.39	0.36	0.36
0.59	0.59	0.56	0.56	0.53	0.53	0.50	0.50	0.47	0.47
0.71	0.71	0.67	0.67	0.64	0.64	0.61	0.61	0.58	0.58
0.82	0.82	0.79	0.79	0.76	0.76	0.73	0.73	0.70	0.70
0.94	0.94	0.91	0.91	0.87	0.87	0.84	0.84	0.81	0.81
				0.99	0.99	0.95	0.95	0.92	0.92

Table D17: Coded Values for Chromosomes 90-94

90	0.777778	91	0.8	92	0.822222	93	0.844444	94	0.866667
Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation
-1.00	-1.00								
-0.89	-0.89	-0.90	-0.90	-0.91	-0.91	-0.92	-0.92	-0.94	-0.94
-0.78	-0.78	-0.79	-0.79	-0.80	-0.80	-0.82	-0.82	-0.83	-0.83
-0.67	-0.67	-0.68	-0.68	-0.70	-0.70	-0.71	-0.71	-0.72	-0.72
-0.56	-0.56	-0.57	-0.57	-0.59	-0.59	-0.60	-0.60	-0.62	-0.62
-0.44	-0.44	-0.46	-0.46	-0.48	-0.48	-0.49	-0.49	-0.51	-0.51
-0.33	-0.33	-0.35	-0.35	-0.37	-0.37	-0.39	-0.39	-0.40	-0.40
-0.22	-0.22	-0.24	-0.24	-0.26	-0.26	-0.28	-0.28	-0.30	-0.30
-0.11	-0.11	-0.13	-0.13	-0.15	-0.15	-0.17	-0.17	-0.19	-0.19
0.00	0.00	-0.02	-0.02	-0.04	-0.04	-0.06	-0.06	-0.09	-0.09
0.11	0.11	0.09	0.09	0.07	0.07	0.04	0.04	0.02	0.02
0.22	0.22	0.20	0.20	0.17	0.17	0.15	0.15	0.13	0.13
0.33	0.33	0.31	0.31	0.28	0.28	0.26	0.26	0.23	0.23
0.44	0.44	0.42	0.42	0.39	0.39	0.37	0.37	0.34	0.34
0.56	0.56	0.53	0.53	0.50	0.50	0.47	0.47	0.45	0.45
0.67	0.67	0.64	0.64	0.61	0.61	0.58	0.58	0.55	0.55
0.78	0.78	0.75	0.75	0.72	0.72	0.69	0.69	0.66	0.66
0.89	0.89	0.86	0.86	0.83	0.83	0.80	0.80	0.77	0.77
1.00	1.00	0.97	0.97	0.93	0.93	0.90	0.90	0.87	0.87
								0.98	0.98

Table D18: Coded Values for Chromosomes 95-99

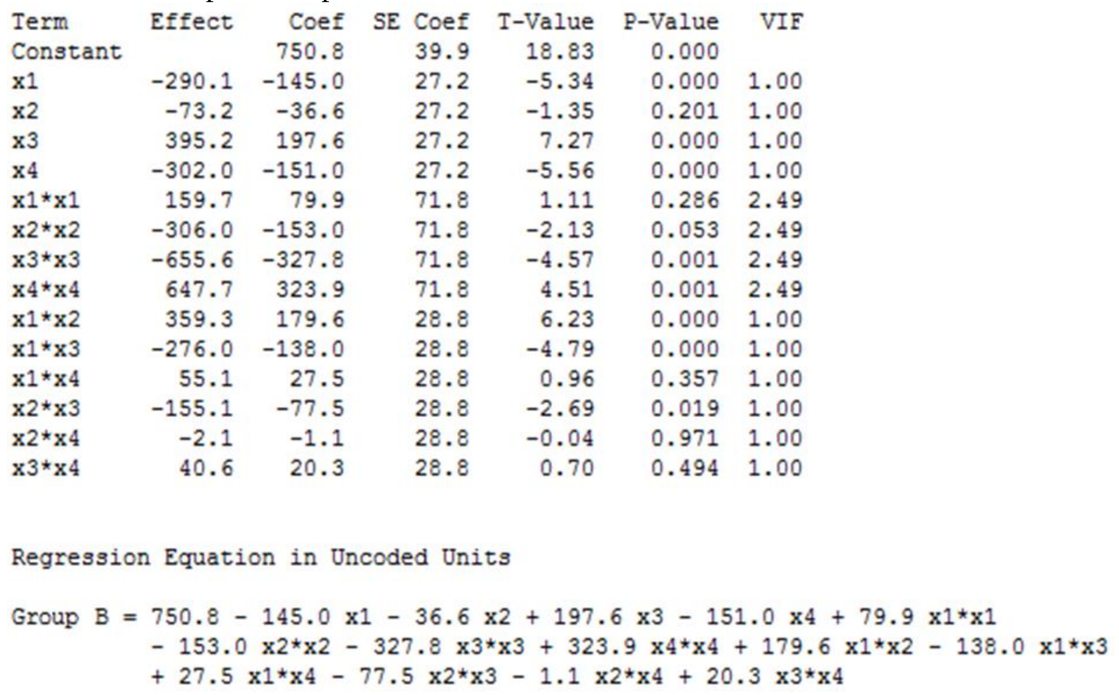
95	0.888889	96	0.911111	97	0.933333	98	0.955556	99	0.977778
Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation	Children	Mutation
-0.95	-0.95	-0.96	-0.96	-0.97	-0.97	-0.98	-0.98	-0.99	-0.99
-0.84	-0.84	-0.85	-0.85	-0.87	-0.87	-0.88	-0.88	-0.89	-0.89
-0.74	-0.74	-0.75	-0.75	-0.76	-0.76	-0.78	-0.78	-0.79	-0.79
-0.63	-0.63	-0.65	-0.65	-0.66	-0.66	-0.67	-0.67	-0.69	-0.69
-0.53	-0.53	-0.54	-0.54	-0.56	-0.56	-0.57	-0.57	-0.59	-0.59
-0.42	-0.42	-0.44	-0.44	-0.45	-0.45	-0.47	-0.47	-0.48	-0.48
-0.32	-0.32	-0.33	-0.33	-0.35	-0.35	-0.37	-0.37	-0.38	-0.38
-0.21	-0.21	-0.23	-0.23	-0.25	-0.25	-0.27	-0.27	-0.28	-0.28
-0.11	-0.11	-0.13	-0.13	-0.14	-0.14	-0.16	-0.16	-0.18	-0.18
0.00	0.00	-0.02	-0.02	-0.04	-0.04	-0.06	-0.06	-0.08	-0.08
0.11	0.11	0.08	0.08	0.06	0.06	0.04	0.04	0.02	0.02
0.21	0.21	0.19	0.19	0.16	0.16	0.14	0.14	0.12	0.12
0.32	0.32	0.29	0.29	0.27	0.27	0.24	0.24	0.22	0.22
0.42	0.42	0.40	0.40	0.37	0.37	0.35	0.35	0.32	0.32
0.53	0.53	0.50	0.50	0.47	0.47	0.45	0.45	0.42	0.42
0.63	0.63	0.60	0.60	0.58	0.58	0.55	0.55	0.53	0.53
0.74	0.74	0.71	0.71	0.68	0.68	0.65	0.65	0.63	0.63
0.84	0.84	0.81	0.81	0.78	0.78	0.76	0.76	0.73	0.73
0.95	0.95	0.92	0.92	0.89	0.89	0.86	0.86	0.83	0.83
				0.99	0.99	0.96	0.96	0.93	0.93

Table D19: Coded Values for Chromosome 100

100	1
Children	Mutation
-1.00	-1.00
-0.90	-0.90
-0.80	-0.80
-0.70	-0.70
-0.60	-0.60
-0.50	-0.50
-0.40	-0.40
-0.30	-0.30
-0.20	-0.20
-0.10	-0.10
0.00	0.00
0.10	0.10
0.20	0.20
0.30	0.30
0.40	0.40
0.50	0.50
0.60	0.60
0.70	0.70
0.80	0.80
0.90	0.90
1.00	1.00

APPENDIX E

Table E1: Group B Complete Model Results



```
Term      Effect    Coef  SE Coef  T-Value  P-Value   VIF
Constant           750.8     39.9    18.83    0.000
x1        -290.1  -145.0     27.2    -5.34    0.000  1.00
x2         -73.2   -36.6     27.2    -1.35    0.201  1.00
x3         395.2   197.6     27.2     7.27    0.000  1.00
x4        -302.0  -151.0     27.2    -5.56    0.000  1.00
x1*x1      159.7    79.9     71.8     1.11    0.286  2.49
x2*x2     -306.0  -153.0     71.8    -2.13    0.053  2.49
x3*x3     -655.6  -327.8     71.8    -4.57    0.001  2.49
x4*x4      647.7   323.9     71.8     4.51    0.001  2.49
x1*x2      359.3   179.6     28.8     6.23    0.000  1.00
x1*x3     -276.0  -138.0     28.8    -4.79    0.000  1.00
x1*x4       55.1    27.5     28.8     0.96    0.357  1.00
x2*x3     -155.1   -77.5     28.8    -2.69    0.019  1.00
x2*x4       -2.1    -1.1     28.8    -0.04    0.971  1.00
x3*x4       40.6    20.3     28.8     0.70    0.494  1.00


Regression Equation in Uncoded Units

Group B = 750.8 - 145.0 x1 - 36.6 x2 + 197.6 x3 - 151.0 x4 + 79.9 x1*x1
          - 153.0 x2*x2 - 327.8 x3*x3 + 323.9 x4*x4 + 179.6 x1*x2 - 138.0 x1*x3
          + 27.5 x1*x4 - 77.5 x2*x3 - 1.1 x2*x4 + 20.3 x3*x4
```

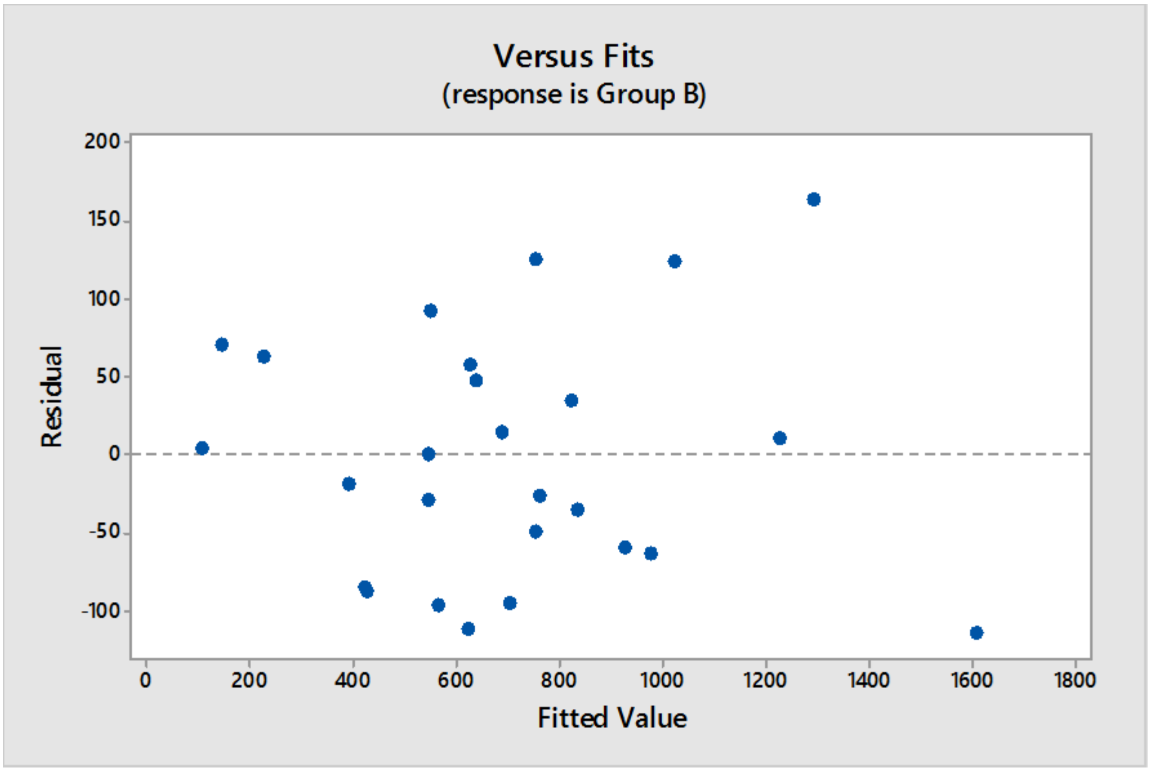


Figure E1: Residuals Scattered Plot of Group B

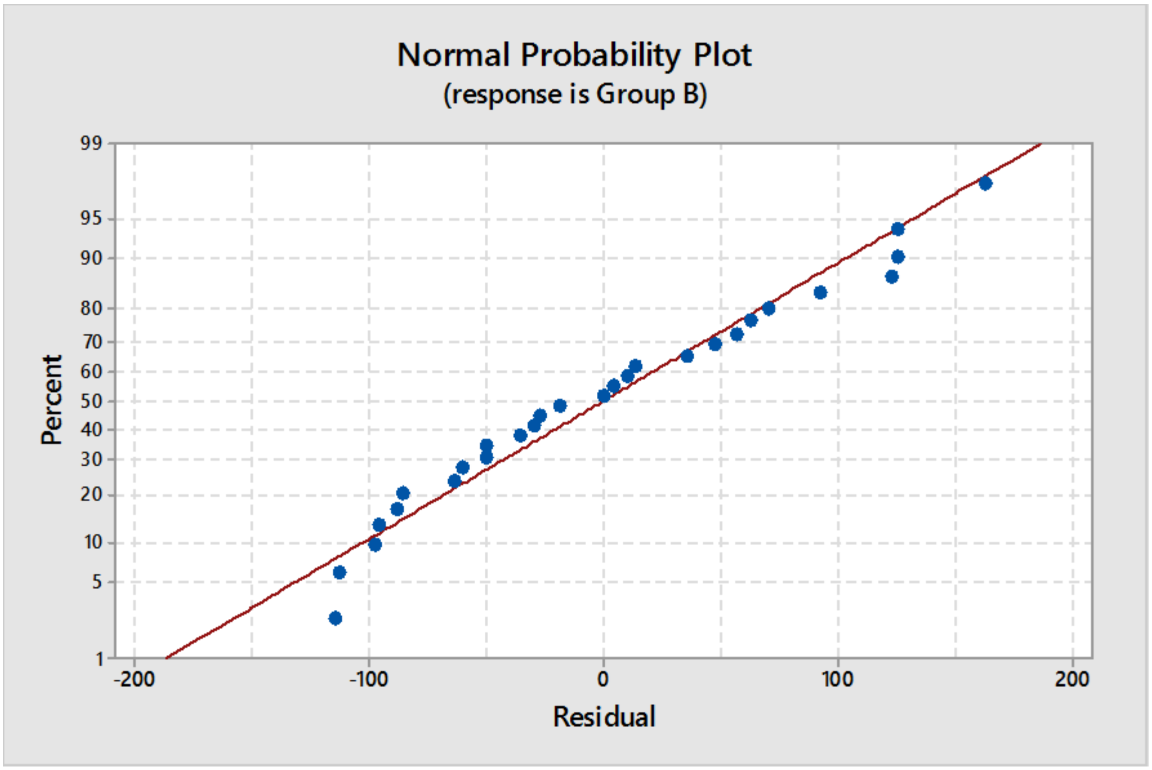


Figure E2: Group B Residuals Normal Distribution

Table E2: Group B Reduced Model Results

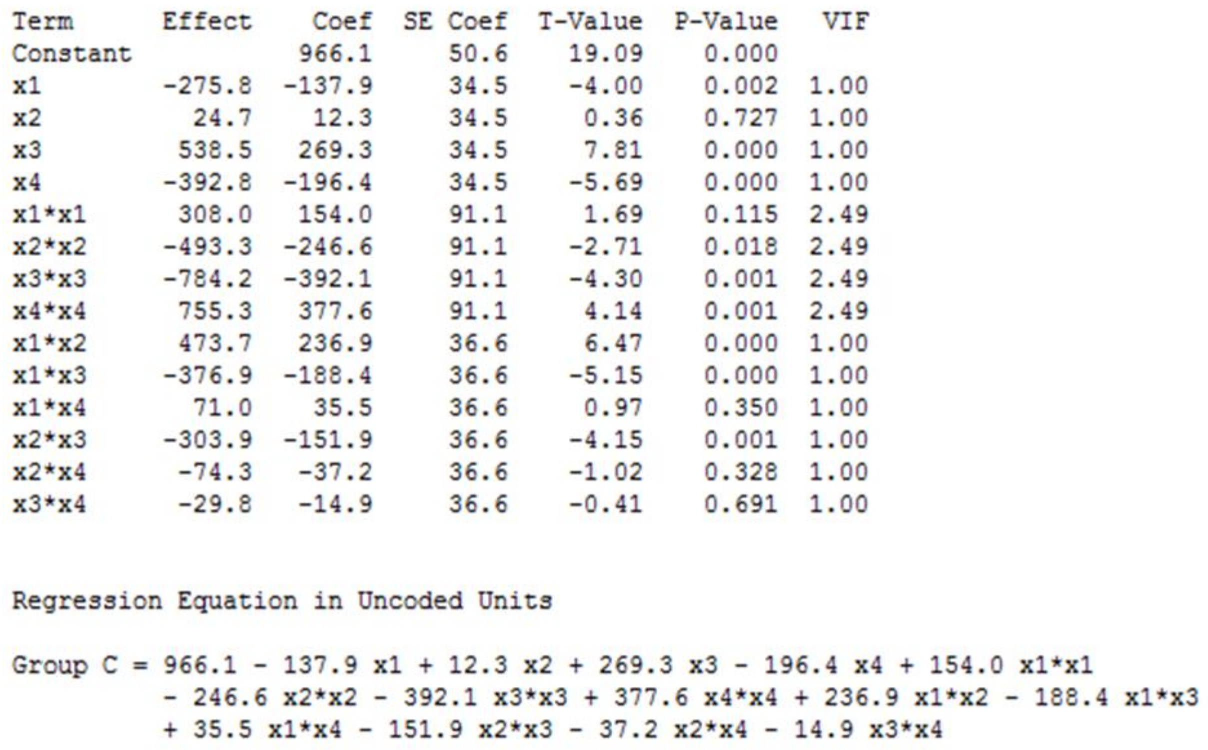
```
Coded Coefficients

Term        Effect     Coef  SE Coef  T-Value  P-Value   VIF
Constant              757.5     37.8    20.03    0.000
x1          -290.1   -145.0     26.1    -5.56    0.000  1.00
x2           -73.2    -36.6     26.1    -1.40    0.178  1.00
x3           395.2    197.6     26.1     7.58    0.000  1.00
x4          -302.0   -151.0     26.1    -5.79    0.000  1.00
x2*x2       -259.7   -129.9     65.9    -1.97    0.065  2.28
x3*x3       -609.4   -304.7     65.9    -4.62    0.000  2.28
x4*x4        694.0    347.0     65.9     5.26    0.000  2.28
x1*x2        359.3    179.6     27.7     6.49    0.000  1.00
x1*x3       -276.0   -138.0     27.7    -4.99    0.000  1.00
x2*x3       -155.1    -77.5     27.7    -2.80    0.012  1.00


Regression Equation in Uncoded Units

Group B = 757.5 - 145.0 x1 - 36.6 x2 + 197.6 x3 - 151.0 x4 - 129.9 x2*x2
          - 304.7 x3*x3 + 347.0 x4*x4 + 179.6 x1*x2 - 138.0 x1*x3 - 77.5 x2*x3
```

Table E3: Group C Complete Model Results



```
Term      Effect    Coef  SE Coef  T-Value  P-Value   VIF
Constant           966.1     50.6    19.09    0.000
x1        -275.8  -137.9     34.5    -4.00    0.002  1.00
x2          24.7    12.3     34.5     0.36    0.727  1.00
x3         538.5   269.3     34.5     7.81    0.000  1.00
x4        -392.8  -196.4     34.5    -5.69    0.000  1.00
x1*x1      308.0   154.0     91.1     1.69    0.115  2.49
x2*x2     -493.3  -246.6     91.1    -2.71    0.018  2.49
x3*x3     -784.2  -392.1     91.1    -4.30    0.001  2.49
x4*x4      755.3   377.6     91.1     4.14    0.001  2.49
x1*x2      473.7   236.9     36.6     6.47    0.000  1.00
x1*x3     -376.9  -188.4     36.6    -5.15    0.000  1.00
x1*x4       71.0    35.5     36.6     0.97    0.350  1.00
x2*x3     -303.9  -151.9     36.6    -4.15    0.001  1.00
x2*x4      -74.3   -37.2     36.6    -1.02    0.328  1.00
x3*x4      -29.8   -14.9     36.6    -0.41    0.691  1.00


Regression Equation in Uncoded Units

Group C = 966.1 - 137.9 x1 + 12.3 x2 + 269.3 x3 - 196.4 x4 + 154.0 x1*x1
          - 246.6 x2*x2 - 392.1 x3*x3 + 377.6 x4*x4 + 236.9 x1*x2 - 188.4 x1*x3
          + 35.5 x1*x4 - 151.9 x2*x3 - 37.2 x2*x4 - 14.9 x3*x4
```

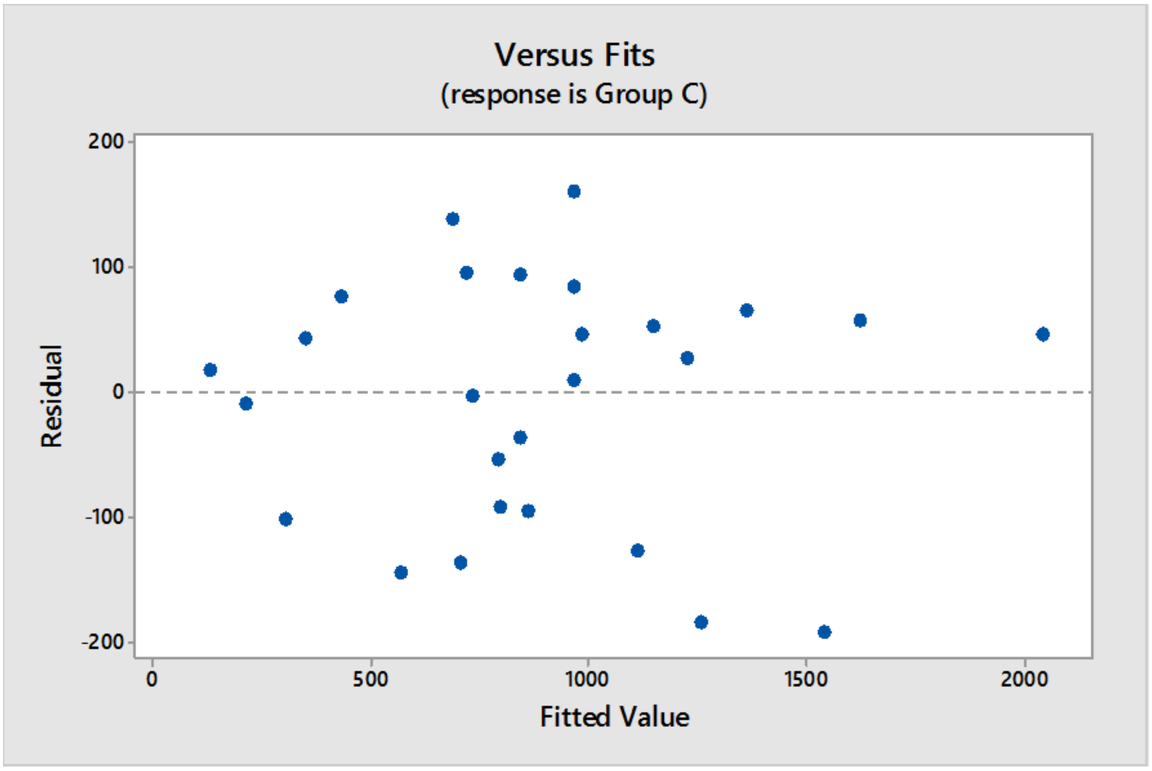


Figure E3: Residuals Scattered Plot of Group C

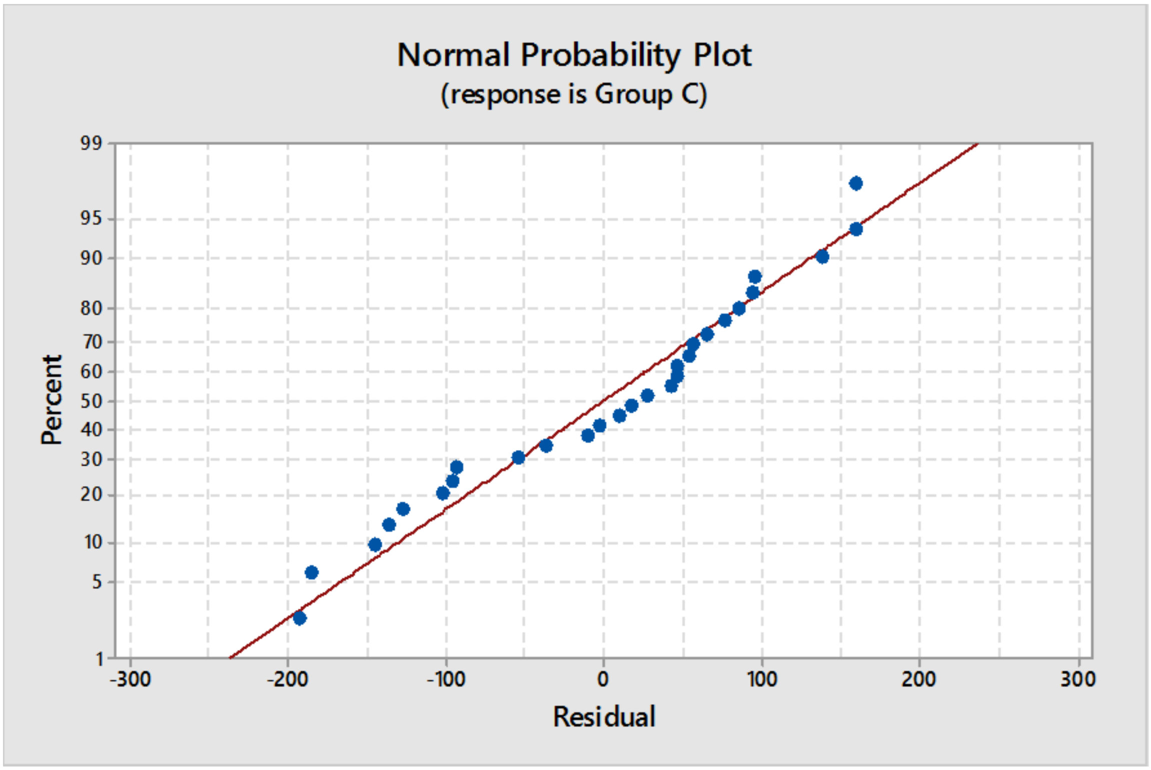


Figure E4: Group C Residuals Normal Distribution

Table E4: Group C Reduced Model Results

```
Coded Coefficients

Term      Effect    Coef  SE Coef  T-Value  P-Value   VIF
Constant           979.0     51.5    19.03    0.000
x1        -275.8  -137.9     35.5    -3.89    0.001  1.00
x2          24.7    12.3     35.5     0.35    0.733  1.00
x3         538.5   269.3     35.5     7.59    0.000  1.00
x4        -392.8  -196.4     35.5    -5.53    0.000  1.00
x2*x2     -404.0  -202.0     89.7    -2.25    0.038  2.28
x3*x3     -694.9  -347.5     89.7    -3.87    0.001  2.28
x4*x4      844.5   422.3     89.7     4.71    0.000  2.28
x1*x2      473.7   236.9     37.6     6.29    0.000  1.00
x1*x3     -376.8  -188.4     37.6    -5.01    0.000  1.00
x2*x3     -303.9  -151.9     37.6    -4.04    0.001  1.00


Regression Equation in Uncoded Units

Group C = 979.0 - 137.9 x1 + 12.3 x2 + 269.3 x3 - 196.4 x4 - 202.0 x2*x2
          - 347.5 x3*x3 + 422.3 x4*x4 + 236.9 x1*x2 - 188.4 x1*x3 - 151.9 x2*x3
```

APPENDIX F

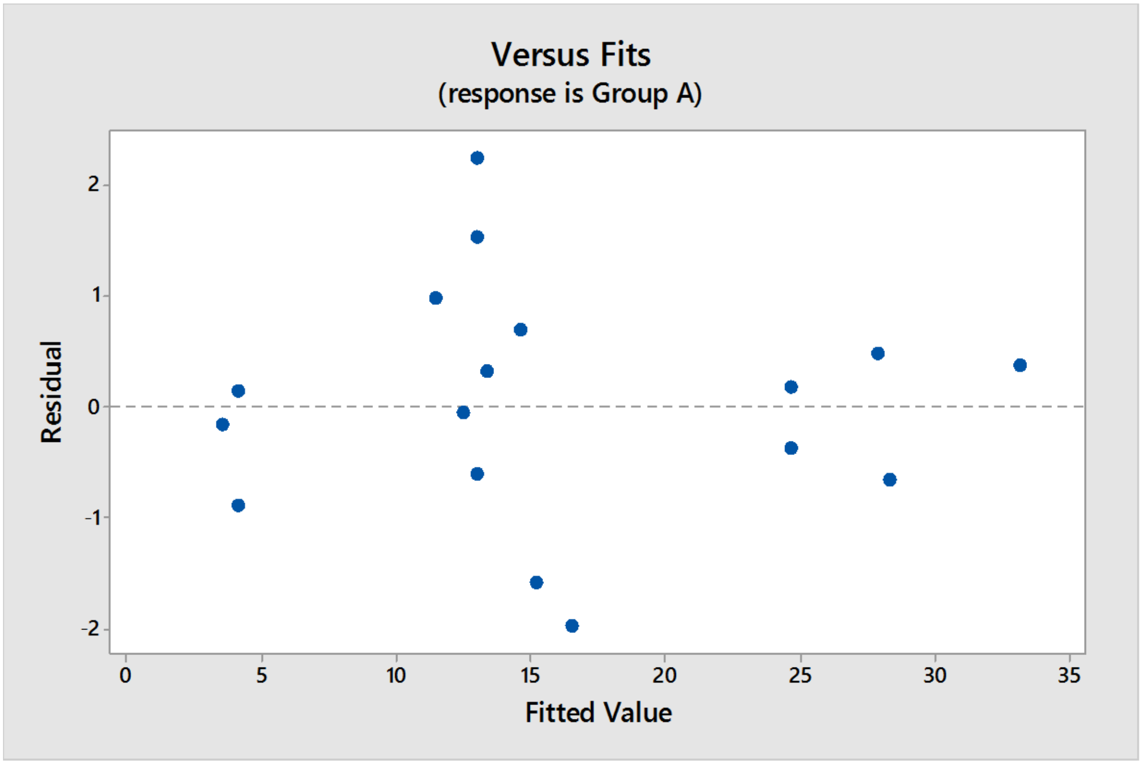


Figure F1: Residuals Scattered Plot of Group A

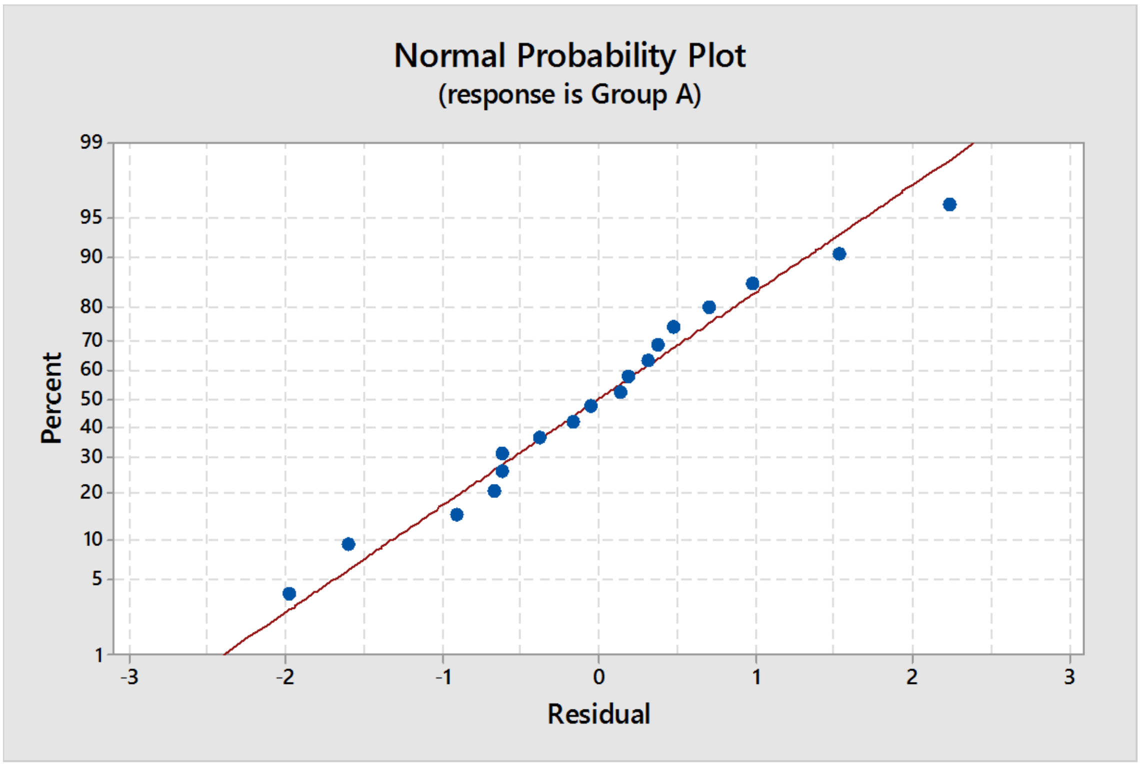


Figure F2: Group A Residuals Normal Distribution

Table F1: Group B Complete Model Results

Term	Effect	Coef	SE Coef	T-Value	P-Value	VIF
Constant		537.4	55.5	9.68	0.000	
x1	114.2	57.1	44.6	1.28	0.236	1.00
x2	-189.4	-94.7	44.6	-2.12	0.067	1.00
x3	919.8	459.9	44.6	10.31	0.000	1.00
x1*x1	-527.4	-263.7	85.7	-3.08	0.015	1.64
x2*x2	116.8	58.4	85.7	0.68	0.515	1.64
x3*x3	370.8	185.4	85.7	2.16	0.062	1.64
x1*x2	-47.1	-23.6	49.9	-0.47	0.649	1.00
x1*x3	136.0	68.0	49.9	1.36	0.210	1.00
x2*x3	-123.1	-61.6	49.9	-1.23	0.252	1.00

```
Regression Equation in Uncoded Units

Group B = -697 + 52.9 x1 - 12.8 x2 - 1.20 x3 - 0.659 x1*x1 + 0.72 x2*x2 + 0.0915 x3*x3
          - 0.131 x1*x2 + 0.0756 x1*x3 - 0.152 x2*x3
```

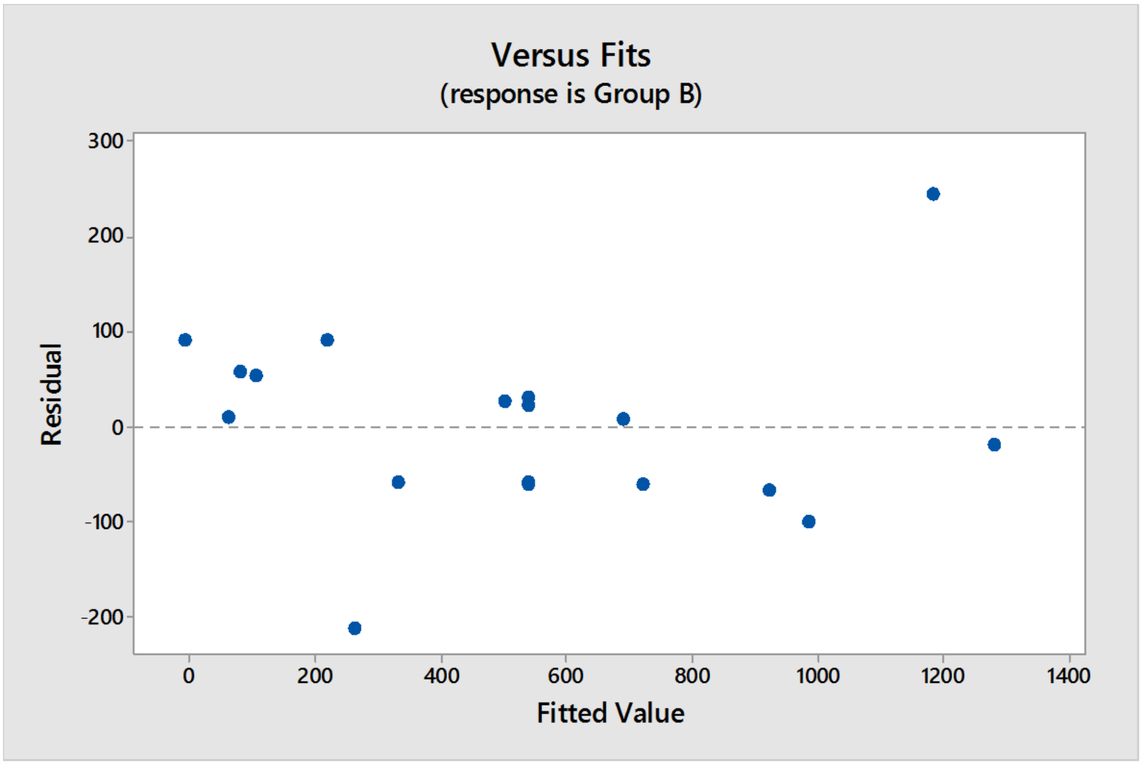


Figure F3: Residuals Scattered Plot of Group B

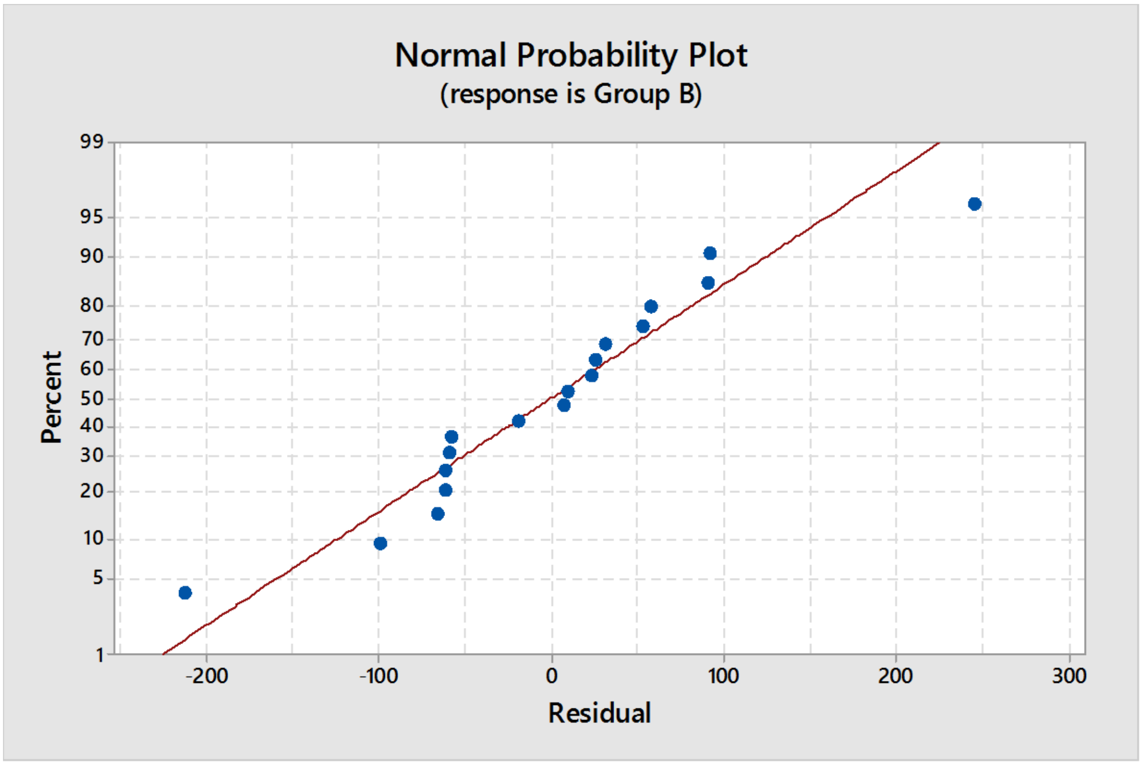


Figure F4: Group B Residuals Normal Distribution

Table F2: Group B Reduced Model Results

```
Coded Coefficients

Term        Effect     Coef  SE Coef  T-Value  P-Value   VIF
Constant             546.8     53.9    10.14    0.000
x1           114.2    57.1     44.7     1.28    0.226  1.00
x2          -189.4   -94.7     44.7    -2.12    0.056  1.00
x3           919.8   459.9     44.7    10.28    0.000  1.00
x1*x1       -486.0  -243.0     80.4    -3.02    0.011  1.43
x3*x3        412.2   206.1     80.4     2.56    0.025  1.43


Regression Equation in Uncoded Units

Group B = 546.8 + 57.1 x1 - 94.7 x2 + 459.9 x3 - 243.0 x1*x1 + 206.1 x3*x3
```

Table F3: Group C Complete Model Results

Term	Effect	Coef	SE Coef	T-Value	P-Value	VIF
Constant		516.1	16.5	31.31	0.000	
x1	-6.9	-3.4	13.3	-0.26	0.803	1.00
x2	-309.2	-154.6	13.3	-11.67	0.000	1.00
x3	597.5	298.8	13.3	22.54	0.000	1.00
x1*x1	-480.7	-240.3	25.5	-9.44	0.000	1.64
x2*x2	75.3	37.6	25.5	1.48	0.178	1.64
x3*x3	70.2	35.1	25.5	1.38	0.206	1.64
x1*x2	-9.5	-4.8	14.8	-0.32	0.756	1.00
x1*x3	-30.0	-15.0	14.8	-1.01	0.341	1.00
x2*x3	-330.6	-165.3	14.8	-11.16	0.000	1.00

```
Regression Equation in Uncoded Units

Group C = -801.1 + 49.10 x1 - 3.89 x2 + 9.89 x3 - 0.6008 x1*x1 + 0.465 x2*x2 + 0.0173 x3*x3
          - 0.0265 x1*x2 - 0.0167 x1*x3 - 0.4081 x2*x3
```

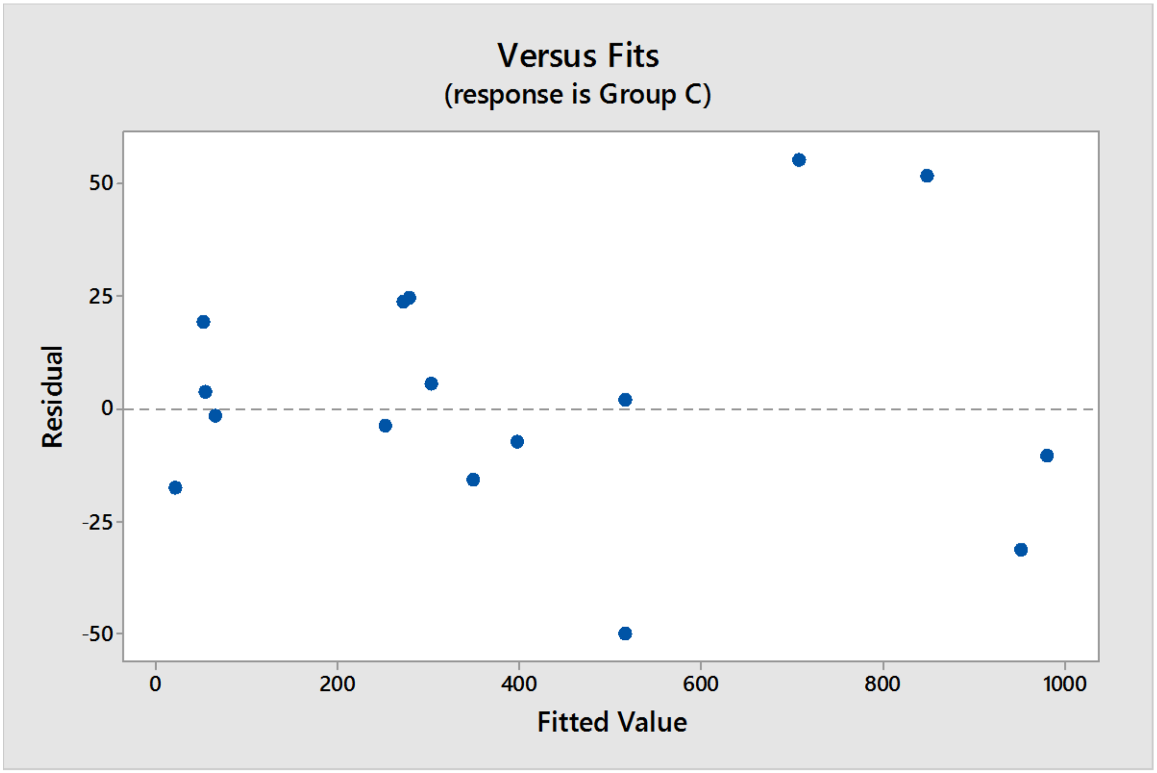


Figure F5: Residuals Scattered Plot of Group C

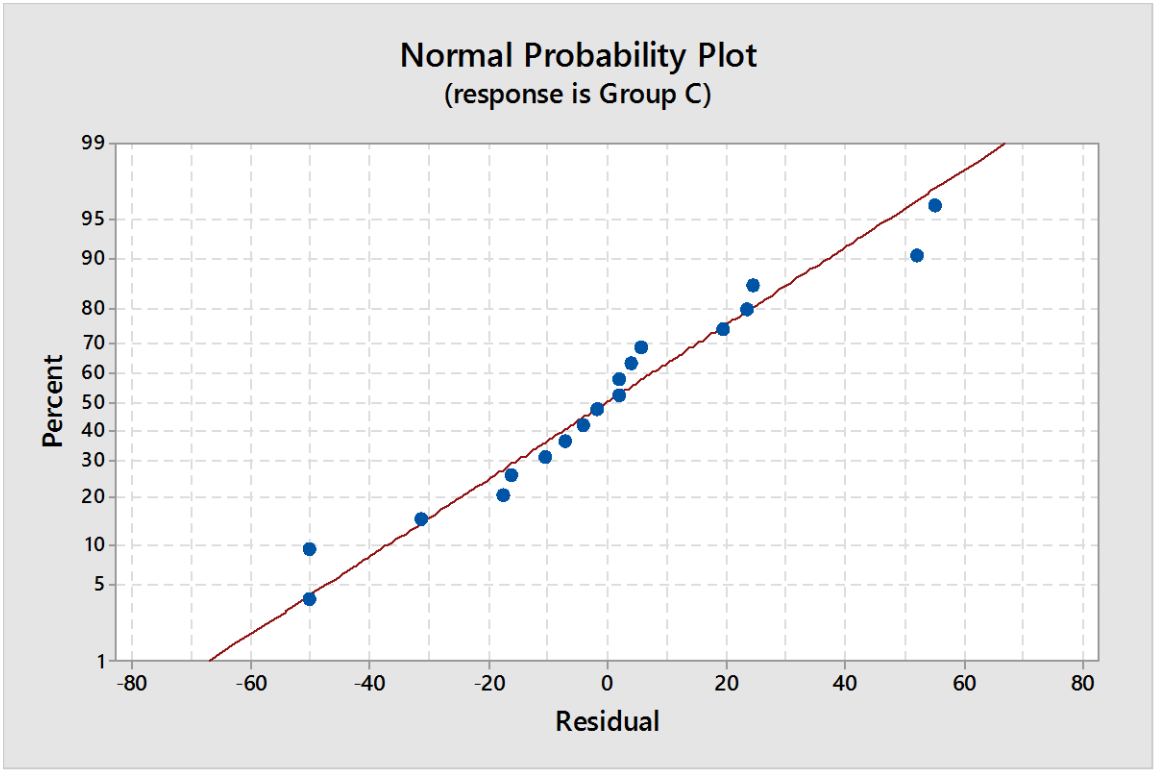


Figure F6: Group C Residuals Normal Distribution

Table F4: Group C Reduced Model Results

```
Coded Coefficients

Term        Effect    Coef  SE Coef  T-Value  P-Value   VIF
Constant             533.2     82.1     6.50    0.000
x1           -35.4   -17.7     73.4    -0.24    0.814  1.00
x2          -303.6  -151.8     73.4    -2.07    0.061  1.00
x3           670.5   335.2     73.4     4.57    0.001  1.00
x1*x1         -386    -193      110    -1.75    0.105  1.00
x2*x3       -410.0  -205.0     82.1    -2.50    0.028  1.00


Regression Equation in Uncoded Units

Group C = 533.2 - 17.7 x1 - 151.8 x2 + 335.2 x3 - 193 x1*x1 - 205.0 x2*x3
```