MECN 7052 – RESEARCH PROJECT

Line Balancing Using Metaheuristic Methods in BMW South Africa

Richard Hart- 0709761P

Faculty of Engineering and the Built Environment

Private Bag 3, Wits 2050, South Africa * Telephone (011) 717 – 7007 * Fax: (011) 717 7009 * Email: febe.pg@wits.ac.za

PLAGIARISM DECLARATION TO BE SIGNED BY ALL HIGHER DEGREE STUDENTS

SENATE PLAGIARISM POLICY: APPENDIX ONE

I _____ (Student number: _____) am a student registered for the degree of _____ in the academic year _____.

I hereby declare the following:

- I am aware that plagiarism (the use of someone else's work without their permission and/or without acknowledging the original source) is wrong.
- I confirm that the work submitted for assessment for the above degree is my own unaided work except where I have explicitly indicated otherwise.
- I have followed the required conventions in referencing the thoughts and ideas of others.
- I understand that the University of the Witwatersrand may take disciplinary action against me if there is a belief that this in not my own unaided work or that I have failed to acknowledge the source of the ideas or words in my writing.

Signature: _____ Date: _____

1 Acknowledgements

I would like to extend my thanks to the following people who provided support to make the completion of this project possible. Dr Campbell who provided direction and advice to me during the course of the project. My manager, Dave Lee, who supported my studies and made it possible to do such a study in BMW South Africa. Finally, to my partner, Kristen Helberg, whose support made all this possible.

2 Abstract

This study documents a project to investigate the possibility of achieving savings in BMW South Africa's Rosslyn assembly plant through the use of metaheuristics to optimise line balancing methods. Through this project, a customised Ant Colony Optimisation algorithm was developed for the optimisation of the frontend assembly line in this plant. This algorithm is one which was designed to take into account many of the constraints which are found in an automotive manufacturing environment such as work areas, shared processes and sequence constraints. Through the use of the algorithm, a solution was developed which shows improvements to the line balancing in the area. These improvements show a 17% reduction in labour costs in the area, an improvement of 13.12% in the area's average work loading and an increase in the average work stability of 17.81%. Additionally, improvements were found which would allow this algorithm to be used in other lines in the assembly plant for further savings and improvements.

3 Table of Contents

4 Table of Figures

5 Table of Tables

6 Introduction

The automotive industry is one which historically has been synonymous with the assembly line since its inception with Henry Ford in the United States in 1913. Assembly lines bring large efficiency benefits to a manufacturing process however they also bring different types of challenges. A dominant concept in the industry is that of continuous improvement. This is a core philosophy in the lean manufacturing principles which have dominated the industry since the success of lean manufacturing in the Toyota Production Group (1). The goal of continuous improvement is to deliver on efficiency and quality improvements which increase the competitiveness of the organisation. The assembly plant of BMW Rosslyn follows this approach, and the process of continuous improvement is supposed to never stop. This study was performed to identify new opportunities for improvements through optimisation of assembly line balancing, and to provide a methodology which can provide improvements on a continuous basis throughout the assembly plant. Metaheuristics were used to find optimised line balancing solutions which showed quantifiable improvements. These can be discussed and implemented in the assembly process. The metaheuristic approach has to take into account what data is available through the company as well as the principles and culture within the assembly plant while creating solutions which are feasible to implement. This is necessary for achieving buy-in from relevant stakeholders in order to realise the improvements which are promised through the study. The goal of this study therefore is to show a solution which provides an improvement in the efficiency and quality of a section of the production process and to obtain a solution which is both implementable by planning structures in the company as well as workable by the operators who are employed to assemble the product.

7 Literature Review

7.1 BMW Plant Rosslyn

BMW's manufacturing plant in Rosslyn, Pretoria was the first manufacturing location to be opened by the BMW Group outside of Germany in 1968 and has manufactured over one million vehicles since this date. The plant encompasses the full production process from body welding and painting to vehicle assembly and has manufactured several models. The model which has been manufactured for most of the plant's lifespan has been the BMW 3-series. Presently the plant is manufacturing the BMW 3-series sedan (F30) and produces over 300 units per day. The plant is scheduled to manufacture the next generation X3 starting in 2018. The competition experienced within this environment is both internal, stemming from the need to match up and improve on group standards and best practices, as well as external from other manufacturers. In this environment the need to continuously improve and innovate is essential if BMW Rosslyn is to remain a key part of the BMW Group's manufacturing network.

The assembly line of the plant is responsible for all processes including fitment of all interior and exterior trim, engine and powertrain as well as initialising and testing of electronic equipment and components. Inclusive of quality inspection processes there are more than 300 processes involved in the manufacture of units on the assembly line per shift over three shifts. This means that the scope of the assembly plant is large and there are many opportunities for optimisation and continuous improvement.

7.2 Assembly Line Introduction

The assembly line is a manufacturing process in which there is a sequence of work stations which are laid out in a sequential, usually linear, layout each with a limited scope of work which contributes to the overall manufacture of the product. The material moves through these stations in a pre-defined sequence and usually with a regular frequency starting at the beginning of the assembly line at a basic or non-existent level of assembly and proceeds down the line being progressively worked on until completion near or at the end of the line. In the stations workers perform regular cyclical work and the work pieces are usually transported between stations through the use of a mechanical transport system which, depending on the size of the product, can vary from simple roller beds to motorised conveyors and material handling systems (2). This methodology can also be called a serial production line due to the manner in which it is laid out and worked on (3). In an assembly line

small repetitive tasks are performed in each station in order to build up the final product with the products moving to the following station when the task is done or moving at a continuous pace in line with what tasks need to be performed (4). The benefits of running a system in this manner is that highly complex processes are broken down into smaller, simpler steps which are easier to learn and manage and also allows for high volumes of products to be produced very rapidly due to the reduction of waste which comes from logistics processes which can be optimised to fit the smaller processes. Therefore this improves the efficiency, cost-effectiveness and productivity of a manufacturing process.

Assembly lines historically were planned for the purpose of producing large numbers of standardised products. However, recent trends have resulted in production lines moving towards smaller volumes of highly customised products. As a result it has become necessary to take into account large amounts of variability in products (5).

The major characteristics which need to be considered are the manner in which the assembly line deals with time and the manner in which the parts move. Some of the types of assembly lines which can be seen in different manufacturing processes are as follows (3):

- Paced Synchronous Production Lines: Where the movement of content between work stations happens in a way that all the jobs index in the station simultaneously, and this is based on a pre-determined cycle time. Therefore the cycle time or process time of every station in the assembly line is the same at any point, because they begin and end their cycles simultaneously. The downside of an assembly line such as this is that it is vulnerable to noise in production processes. Process failures and deviations can cause every process to stop in order for a problem to be resolved. Any deviations in such a line will usually be immediately apparent.
- Un-paced Synchronous Production Lines: A synchronous line which advances only when all the stations have completed their tasks and not at a pre-determined pace. Assembly lines in this format are as susceptible to deviation as a paced synchronous line. However any deviations in the process or equipment may not be readily apparent. This could lead to losses in productivity which are difficult to identify, because a bottleneck may not be readily apparent.
- Asynchronous Production Lines: A line in which the jobs move independently of one another and are timed independently of one another. An assembly line of this type can be difficult to manage as inconsistency between processes can lead to a build-up of work in progress (WIP) inventory around bottlenecks in the line. Additionally, such a line can hide a large amount of waiting time, because processes operate in a manner which pushes parts to the next workstation.

A serial production line and a synchronous production line are operated in order to achieve the flow promoted as part of lean manufacturing practices. This is because they apply a standardised process time throughout the assembly process. In fact, these line types are fundamental concepts for the introduction of lean manufacturing because they are useful for achieving many of the targets which form the core of lean manufacturing methodology. Lean manufacturing is reliant on highly standardised, predictable work.

7.3 Lean Manufacturing

Lean manufacturing is a term which was coined by J. Krafcik in 1988 (6) and refers to a production methodology which was originally developed within the Toyota automotive group in order to remain competitive with international competition. Lean manufacturing can be seen as an approach which has the primary focus of eliminating waste. This waste can be in several forms: non-value adding work ("*muda*"), overburden ("*muri*") and unevenness ("*mura*"). The goal is to achieve a production system which flows smoothly with a high level of quality, while minimising the amount of resources required to produce the product. In doing so, efficiency is maximised and therefore the cost-effectiveness of producing a product is improved. The most commonly known aspect of lean manufacturing is that of the reduction of *muda.* One of the first things that is taught in lean manufacturing is the seven types of waste. Reduction of waste can maximise the proportion of value created for the end customer compared to the amount of time spent on non-value adding activities (1). The original seven types of waste are as follows, although they may vary in some references:

- Transport
- Inventory
- Motion
- Waiting
- Overproduction
- Over Processing
- Defects

Targeting these forms of non-value adding work, efficiency can be increased. However, this must not compromise the principles of *muri* and *mura*. Non-value adding processes can be found in every part of an organisation and at different levels, and is not restricted to assembly line processes.

Leading on from this is the necessity to define exactly what the customer wants. Anything which falls outside this scope should form part of one of the seven wastes and therefore will need to be reduced/eliminated. This is done by determining what

steps in a production process unambiguously add value. This can vary depending on the industry and product in which lean methodology is being applied, because different customers have different wants (1). A loose definition of value adding content used in automotive circles is anything which changes the fit, form or function of a product which will not be undone before reaching the customer.

According to Womack *et al.* (1) several important steps for the implementation of lean manufacturing are the following:

- Identify the *value stream* for each product – look at the creation of the value from the start to the finish
- Make value *flow* without interruptions – Move the product continuously from start to finish without batch thinking, thereby increasing efficiency and decreasing lead time.
- Let the customer *pull* value from the producer – This allows the system to respond to demand and customer wants to produce the right amount of products with minimal waste.
- Pursue *perfection* – Reducing waste and increasing efficiency is something which continues.

Inherent in this understanding is that the system must have high efficiency while flowing without any form of disruption and interruption. It must also be flexible in order to adapt to what customers want without disruption while producing only what the customer wants and will pay for. The final point leads directly to the core lean principle of continuous improvement. This is a mind-set where improvement is always sought after at all times, by all members of the organisation, and more specifically, the forms of waste previously mentioned should always be reduced where possible (7).

7.4 Line Balancing

In a lean manufacturing environment within a paced assembly line it is important to realise that a customer does not only want certain criteria in a product but also demands only a certain amount of the product. Therefore another concept which is integral to a lean production system, particularly in an assembly line manufacturing process, is that of Takt time. A definition of takt time is “The desired time between units of production output, synchronized to customer demand (8).” The principle is that, in order to match customer demand, a specific amount of a product needs to be produced. Therefore the assembly line should be paced to match this to avoid the waste of overproduction. This is also integral to the concept of line balancing as this provides the target or maximum point to which a work package can feasibly be loaded. If a work package is loaded to the takt time without being overloaded and is

primarily doing value-added content, then it could be said that it is running at a high level of efficiency.

Line balancing is a method which is used to balance the work load within a production environment. It is primarily used in an assembly line process due to the consistent cycle times and the small consistent processes which the manufacture of a product is broken into. The purpose of this is to avoid waiting times, overproduction as well as overloading (*muri*). Waiting and overproduction are forms of waste which affect a production line's efficiency and therefore increase the costs of production. Overloading is a state where a process in an assembly process has too much work to be performed within the available time. This overloading can lead to overtime and high expediting costs to keep up and/or unsatisfied customers due to poor quality or failure to meet delivery targets (9). The two requirements for an effective quantitative line balancing approach are a set Takt time and time study results.

Before line balancing is performed it is necessary to analyse the work content which needs to be completed using standardised time measurement techniques. This is required as any line balancing approach needs to be quantified. If it is not quantified, then subjectivity of the analyst and variation between processes and workers could lead to the line balancing being inefficient or unbalanced. Variations in measurement could also cause the time study to be called into question which can lead to a complete failure of the method. Therefore an approach for time study recommended by Groover (10) is as follows:

1. Define the standard time study method.
2. Divide the process into smaller steps or work elements.
3. Time the work elements using the standard method.
4. Evaluate the speed at which the specific worker completes the task
5. Increase the time by a fixed margin to allow for errors and variation in the study.

The most essential part of this process is that it must be standardised in order to avoid as much variation as possible. Higher amounts of standardisation and lower amounts of human interpretation in time studies lead to more reliable results in line balancing.

The largest constraint on any line balancing problem is build sequence. This can cause line balancing problems to be difficult to solve, since it can lead to many solutions being infeasible. It is important to take all sequence constraints into consideration. It is useful to map out the build sequence in a graphical manner. An example of a 12 task problem can be seen in Figure 1 (4).

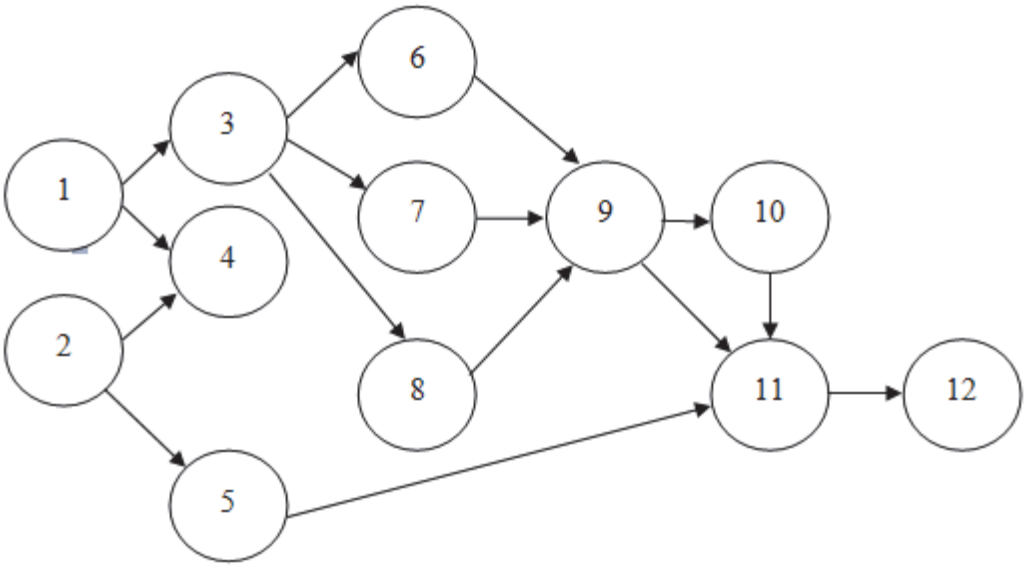


Figure 1: Example of a 12 Task Sequence Diagram

In the diagram above, the sequence represented needs to be adhered to where a task can only be performed if all proceeding tasks have been completed. Skipping a task will either make subsequent tasks impossible or will obstruct and render earlier tasks impossible. Other constraints may be present in a line balancing problem relating either to the movement of workers or to structural and equipment related constraints. It is necessary for the design of an assembly line and therefore line balancing to consider the cost and reliability of the system, complexity of the tasks, equipment selection, assembly line operating criteria, different constraints, scheduling, station allocation, inventory control and buffer allocation (5).

7.4.1 Line Balancing on a Two-Sided Assembly Line

A two-sided assembly line is one which is typically used to assemble large, high-volume products. The key characteristic of an assembly line of this type is that the workers can perform tasks on two or more sides of the product and therefore, in order to maximise efficiency, the processes are often designed to be split by sides of the product. This avoids a large amount of non-value adding time travelling to multiple points of a product. This line balancing problem is effective in representing problems in which tasks have a specific sequence in which they need to be performed.

An effective way to represent a two-sided assembly line problem is a network diagram proposed by Kim *et al.* (11). This network diagram, while similar to the one previously shown, not only represents the sequence of work but also the amount of time it takes and the area of the unit in which the work should be performed. The area of work is essential when considering larger products such as those seen in automobile manufacturing.

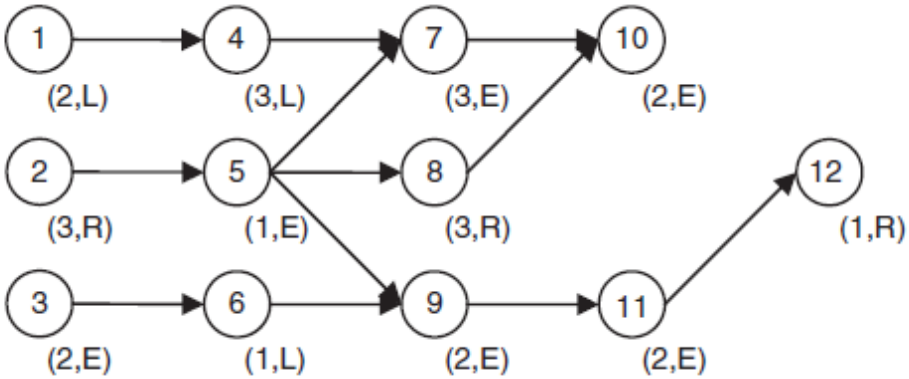


Figure 2: Two-Sided Assembly Line Network Diagram

Figure 2 represents a 12 task problem with a build sequence illustrated by the connectors between tasks and in the field below the time for the completion of the task followed by the area in which the task should be performed. This process can be done by four operators in two stations without crossing between areas of the product as can be seen in Figure 3.

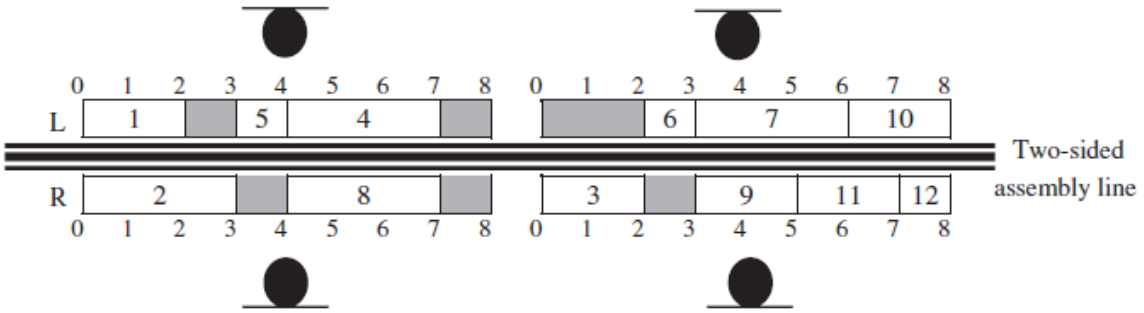


Figure 3: Potential Solution for the Two Sided Line Balancing Problem

Build sequence plays an even more important role in a two-sided line, since there may be idle times due to an operator waiting for another operator to complete a task. This could be a result of a requirement for tasks to be worked on by more than one operator or if the assembly line is un-paced and operators will be waiting for counterparts on other areas of the product. A two sided assembly line has the

benefit of saving a significant amount of space and allows larger products which may require work to be done on both sides of a line to be built. Additionally a two-sided production line layout can be an effective way to reduce a large amount of movement around the product, and this can help to counteract idle time losses. The downside of a two-sided assembly line is that it adds an additional layer of complexity to the line balancing problem.

7.4.2 KPI's for Line Balancing Evaluation

It is essential when performing line balancing to determine measures which can be used to calculate the performance of a particular line balancing solution. There are many KPI's which can be considered when looking at a line balancing problem, some recommended by Uddin *et al.* (5) are higher efficiency, less balance delay, smooth production, optimised processing time, cost effectiveness, overall labour efficiency and just in time production (JIT).Despite the number of criteria which can be found in literature it is essential that KPI's are developed which can be quantified such as the KPI's recommended by *Grzechca* (4):

- **Line Efficiency:** This measure shows the average utilisation of the entire line in a percentage format.

$$\boldsymbol{LE = \frac{\sum_{i=1}^{K} ST_i}{c \cdot K} \cdot 100\%}$$

  Where *K* is the total number of workstations, *c* is the cycle time and ST_i is the station time of station *i*.

- **Smoothness Index:** This measure aims to describe the relative smoothness for a line balance with a perfect balance being indicated by 0.

$$\boldsymbol{SI = \sqrt{\sum_{i=1}^{K} (ST_{max} - ST_i)^2}}$$

  Where ST_{max} is the maximum station time.

- **Time of the Line:** This measure calculates the amount of time which is needed for the product to be completed on the assembly line.

$$\boldsymbol{LT = c \cdot (K - 1) + T_K}$$

Where T_K is the processing time of the last station.

These KPI's provide an indication of the effectiveness of a line balancing with regard to efficiency, balance and lead time. Efficiency is measured through the line efficiency measure by showing how well resources are being utilised. The balance of the line, referring to how equally each station is loaded, is measured by the smoothness index as this measures the variability between all stations. Lead time, which is the amount a product takes to pass through the assembly line, is measured directly in the time of the line calculation.

7.5 Operations Research Approach

In many organisations the level of complexity found in day-to-day processes are requiring increasingly higher manpower capacities and specialisations. Due to this it is common to find that many organisations use automation to physically manufacture products as well as optimise and run processes and background tasks. The use of automation can allow specialists to focus on implementing changes and optimising existing processes rather than spending significant amounts of time maintaining and running existing processes.

Operations research methods, such as linear and non-linear optimisation, come in many forms. There is a vast body of research into these methods that provides a platform to automate many optimisation processes. There is a common approach taken when formulating and solving problems in this field in order to provide a structured and logical methodology. The most common modelling approach is as follows (12):

1. **Defining the problem and gathering data:** This step involves studying the system in order to get the necessary detail as well as to make a well-defined problem statement. This problem statement needs to be structured in such a way that it is as specific as possible while encompassing the goals of the involved parties (the owners, the employees, the customers, the suppliers and the government) and being consistent with the goals of the organisation.
2. **Formulating the mathematical model:** It is necessary in this phase to describe the problem as a model made up of decision variables, an objective function, constraints and parameters which represents the essence of the problem and defines what the problem should optimise as well as what can and cannot be done. The objective function should measure the overall performance of the algorithm in order to achieve a solution which is in line with the goals of the organisation.
3. **Deriving solutions from the model:** After a mathematical model has been formulated a procedure needs to be developed in order to derive a solution for

the problem. There are a large number of procedures which have been researched and documented which can be used to solve many different types of problems in this process. The goal here should be to look for an optimal solution or, failing this, a solution which achieves the goals of the organisation and adds value to the organisation. The method used to solve the problem can determine the quality of the solution or whether a solution can be found at all.

4. **Testing the model:** In this step the model is tested in order to look for bugs and flaws and is re-run until sufficiently valid results are found which are consistent with how the real-life process would be expected to behave. It is important to document the changes in order to see the changes that were made to the model. A part of this step would be to do a post-optimality analysis in order to determine weaknesses to the model and to identify what would happen to changes in the model.
5. **Preparing to apply the model:** This step involves documentation systems and developing additional software to allow the solution to be used in a business environment.
6. **Implementation:** The model is introduced to the company through careful explanation. The accuracy must be confirmed and it is important to have management support at this point. Procedures on use are put in place and standardised.

When running an operations research study it becomes important, in the step of deriving solutions to the model, to use an appropriate method for the solution to the model. This is largely determined by the model which is formulated as different methods suit different problem formats. There are many types of problems, all with different characteristics.

7.6 Line Balancing using Operations Research Methods

The line balancing problem is one which is frequently considered in manufacturing environments. They can be complex problems, particularly if the assembly lines has hundreds of process steps. There can be very different variants of the problem based on the structure of the assembly process and the targets which the problem aims to achieve. The targets are largely based on number of stations, cycle time and smoothness as discussed by Uddin *et al.* (5):

1. A problem which minimises the number of stations while the cycle time remains constant.
2. A problem which minimises the cycle time while the number of stations remain constant.
3. A problem which minimises both the cycle time and number of stations.

4. A problem which maximises the work smoothness and/or work relatedness.

These variants affect how the objective function is structured and calculated; therefore what the best solutions are.

When considering the structure of the process itself, two considerations are the model allocation on the assembly line and the working structure of processes on the line. The model allocation can vary between single model, multi model in batches and intermixed multi model processes, with a corresponding increase in complexity. The structure varies in a manner where there can be parallel stations, U shaped lines and two sided lines.

Within these varying types of problems there are many different approaches which can be taken to solve the problem:

- **Manual Line Balancing:** This approach involves a user manually changing processes and work stations in order to try and achieve an effective balancing. This approach requires a large time investment and may not find an optimal solution. It can be done with minimal resources other than time. It is essential that the person doing the line balancing has an intimate knowledge of the processes.
- **Line Balancing with Simulation:** Simulation can be used to aid the line balancing process by providing metrics which can be used to guide the balancing process. This requires more resources in terms of software and expertise and can take a large amount of time to effectively model the problem.
- **Heuristic Line Balancing:** This approach involves creating a model which can search for a high quality solution based on a random search methodology in order to search across multiple feasible solutions. This requires significant computing resources and requires a large amount of expertise in model development as well as time for model development. It can provide long-term savings if an effective model is built which can solve multiple problems.

Each one of these approaches have advantages and disadvantages. Manual line balancing is an approach which is accessible to many in an assembly environment and is easily trainable. It will seldom provide an optimal solution and can be easily manipulated. Using simulation can provide accurate metrics, but decision making will still need to be done manually or with a heuristic approach in order to feed inputs into a simulation model. It can also greatly increase the lead time of line balancing. Heuristic line balancing requires a significant expertise to model and test as knowledge of heuristic methods is required. If it is done effectively it can provide the most efficient solution for a line balancing problem (11).

7.6.1 Metaheuristics

When searching for an optimal solution to an optimisation problem, there may be cases where the problem being solved is too large to find an optimal solution by testing every feasible solution. While linear programming and other optimisation programming methods are adept at providing optimal solutions, there are cases where such methods will not be able to provide sufficiently accurate solutions or will simply require too many computational resources to solve. This is normally the case when the problem has a large number of variables and/or is highly constrained. This can lead to there being a number of local optima, which can lead to the optimum solution not being found. Standard heuristic methods in particular are vulnerable to local optima (12). Therefore other approaches need to be considered.

Situations such as these are usually found where the type of problem being looked at is a non-linear programming problem. When the problem gets larger, the likelihood of non-linearity increases. With non-linear programming problems it is likely, and in many cases certain, that local optima will exist and therefore the determination of the global optimum becomes challenging, unless a method is used which accounts for this and is able to identify or escape local optima. Local optima are solutions which appear to be optimal when compared to surrounding solutions. However, when the entire solution is considered, there is a more optimal solution to be found. An example of this can be seen in Figure 4 (13).

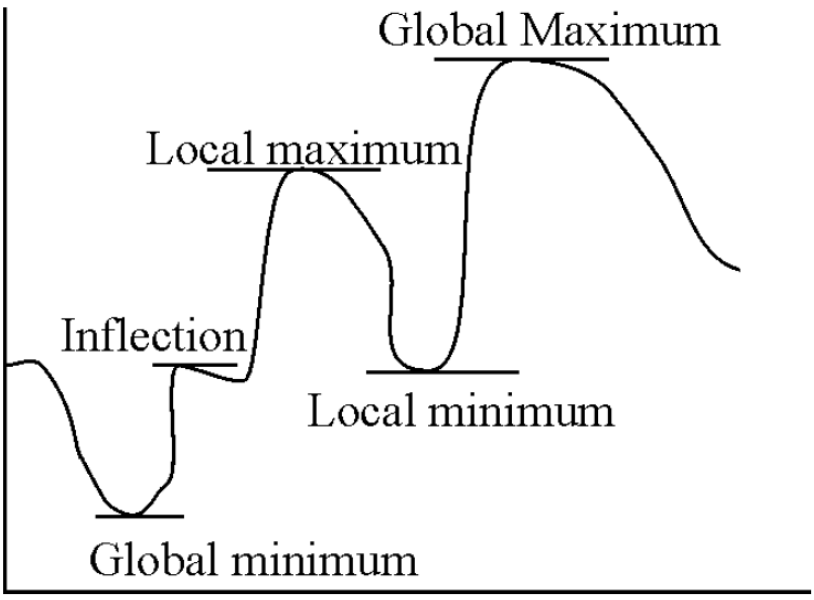


Figure 4: Illustration of Local Optima

In Figure 4 it can be seen that the local maximum and local minimum could be seen as an optimal solution depending on whether the problem is a maximisation or minimisation problem. This would be the case if only the surrounding solutions were compared. However, as shown in Figure 4, there are "better" solutions which are called the global maximum and global minimum, since they are the maximum and minimum for the entire solution space.

The heuristic approach is one which follows a common sense approach. These methods are not always able to find the optimum solution, but rather aim to find a solution which can be considered to be effective enough for implementation (4). Metaheuristics are a type of heuristic method which randomly searches a solution space to find a "best" solution instead of an optimum one. Metaheuristics have come to incorporate a number of features which allow them to avoid local optima. Many metaheuristics do this through searches in the neighbourhood of the current or previous solution(s). This is done by moving between local solutions or by using population-based procedures (14).

The most basic form of meta-heuristic is one of local search, where the algorithm searches for a solution with a better value of the objective function than the current solution. The algorithm does not move on from the current solution until a better one in the neighbourhood is found. The flaw with this type of algorithm is that it has no way of escaping a local optimum and will always determine a local optimum as the best solution (15).

Another type of algorithm is the constructive algorithm. The major difference between this and the local search algorithm is that, while the local search algorithm deals with modifying complete solutions to find better solutions, the constructive algorithm "constructs" a solution by either randomly or through a pre-determined logic choosing the next step to take until a complete solution is developed (15).

7.6.1.1 Genetic Algorithms

The concept of Genetic Algorithms (GA) is one of the oldest forms of optimisation using metaheuristics originating from the 1960's and is based on the ideas of mutation and selection which is central to Neo-Darwinism. Decision variables in genetic algorithms are most commonly represented as binary strings of a uniform length which represent all the decision variables in the problem. The comparison that can be made to evolutionary theory is that a single solution string is equivalent to the genome of a member of the population. The way in which this relates to an optimisation problem is through the concepts of crossover (reproduction) and mutation (16).These methods can be used in many ways as there are many decisions which need to be made when considering the algorithm such as crossover-

AND-mutation or crossover-OR-mutation, the frequency with which the rules are applied and whether they are applied at random locations.

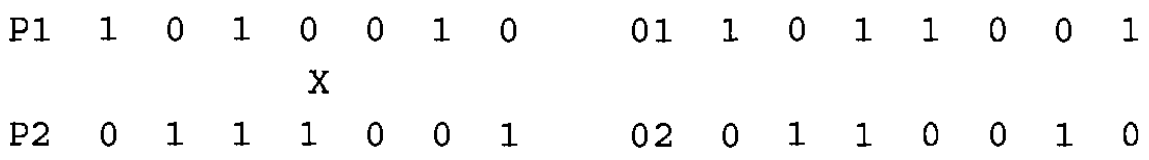


Figure 5: Parent and Offspring Strings in a Genetic Algorithm

The concept of crossover is represented in Figure 5. Two parents exchange parts of their characteristics with the other to produce two offspring solutions with characteristics of both parent solutions. Crossover is most commonly performed on every cycle of the GA. However, it can be performed in different ways. Some of the ways in which it can vary are as follows (16):

- Fixed location versus variable location crossover: In this instance a choice needs to be made whether to crossover on a fixed point of the string or to randomise/select a location in the string to perform a crossover.
- Single point versus multiple location crossover: This requires the decision to be made whether to crossover at a single point or to crossover at multiple points in the algorithm, one frequently used example is a two-point crossover in which two crossover points are selected.
- Frequency of crossover: In most genetic algorithms a crossover is done with every iteration. However, this can vary, depending if mutation is done with or separately from the crossover.

In the case of a problem in which non-repeatability of a value of a decision variable is important, direct crossover is not effective, as it will frequently lead to the repetition which needs to be avoided. In this case an alternative method of crossover will need to be used (16).

Mutation is a concept in which a random piece of the string is occasionally changed to another value based on a pre-determined logic or set of criteria. There are many ways in which the concept of mutation can be applied. The general concept is that it is not applied to the same part of the chromosome but is rather applied randomly in order to prevent the algorithm getting stuck in local optima and to bring variability into the population. While the mutation rate, μ, may vary from problem to problem, the most common method is for the average number of mutations per string, λ, to be equal to one. Therefore the mutation rate needs to be $\mu = \frac{\lambda}{l}$where *l* is the length of any string (16).

7.6.1.2 Tabu Search

Prior to the emergence of metaheuristics, the foremost type of method used for the solution of complex problems were local search methods. The flaw of such methods was that by their nature they terminated when they found a local optimum, which in many cases, was a poor or mediocre solution. Tabu Search was developed from this base as a method which is able to escape local optima by incorporating a memory function. This allows the algorithm to consider solutions which are worse than the previous "best" solution, so as to find a point where the solutions increase in viability again. The search procedure itself works as a local search method which constantly searches for a more optimal solution by changing the current best solution to find a better solution near the current optimum (17).

The memory function of Tabu search broadly works by creating a list of the most recent changes which were made in the solution. These moves are declared "tabu" and are prevented from being implemented again (17). The handling of the list normally runs on a first in, first out basis (FIFO), such as in Figure 6, where the oldest solution (Solution 1) is pushed out because of the newest solution (Solution 5). This allows solution 1 to be considered again, as it is no longer on the tabu list. This causes the algorithm to continuously move in a new direction as it prevents the process from reverting to the previous "best" solution.

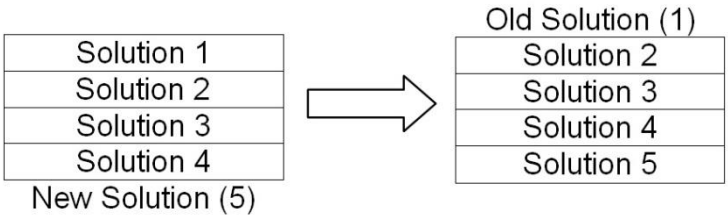


Figure 6: Tabu Search Logic

The only variable which is controllable on a tabu list is that of the length of the list itself, a long list increases the likelihood of escaping local optima, but it can also restrict the local search process by making many solutions unavailable for re-testing and improvement. Some literature (18) suggests that varying the length of the tabu list during the search process can increase the effectiveness in finding a solution by restricting and releasing restrictions on the search as required. This can be done by having a long tabu list initially and gradually decreasing its length as more iterations are tested. This allows the algorithm to test previous best solutions more thoroughly.

An important consideration when using tabu search is the termination criteria. Without this the algorithm could continue searching indefinitely. The most common termination criteria are as follows (17):

- After a certain amount of time or iterations.
- After a certain amount of iterations without an improvement in the objective function being seen.
- When the objective function reaches a pre-defined value or better.

The choice of which termination criteria to use, like the length of the tabu list, depends on the problem being solved, as well as what is required of the algorithm.

7.6.1.3 Simulated Annealing

Simulated Annealing (SA), like Tabu Search, is a variation of a local search method which allows the algorithm to consider less feasible solutions in order to avoid getting stuck in local optima. It does this by emulating the physical process of annealing, where a crystalline solid is heated and then allowed to cool slowly to form a stronger crystalline structure. The algorithm does this is by defining a "temperature" constant t_k, which starts high and decreases as the algorithm progresses and processes more candidate solutions. This constant is used to encourage the algorithm to accept non-improving solutions in order to climb out of local optima. As the constant decreases, the algorithm is less likely to accept non-improving solutions. A common setup is as follows where *w* is the current solution, *w'* is the next, neighbouring solution, *P* is the probability of accepting *w'* and *f(w)* and *f(w')* are the objective function values of the two solutions (19):

$$P = \begin{cases} exp[-(f(w') - f(w))/t_k] & if\ f(w') - f(w) > 0 \\ 1 & if\ f(w') - f(w) \leq 0 \end{cases}$$

Where

$$t_k > 0\ for\ all\ k\ and\ \lim_{k \to \infty} t_k = 0$$

The probability *P* decreases as the algorithm progresses. This leads the algorithm to settle on a solution based on both the current temperature constant as well as the relative quality of the current solution. The algorithm rapidly searches the sample space initially, while later exploring the better solutions for smaller, local improvements.

7.6.1.4 Ant Colony Optimisation

Ant Colony Optimisation (ACO) is a form of meta-heuristic which is inspired by the natural phenomenon of how ants find the shortest path between two points over time. ACO is a constructive metaheuristic meaning that it builds a full solution based on information provided to the heuristic. What makes ACO different to a basic constructive algorithm is the addition of a memory function. This memory function is modelled on the manner in which ants leave pheromones for other ants to follow. In the algorithm, artificial "ants" construct solutions by selecting the next point to move to in a step by step fashion. They make these decisions based on the full set of constraints, as well as what previous ants have done, via the pheromone levels which have been recorded. The decision that an ant makes is based on a probability which is determined by:

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in \mathcal{N}_i^k}[\tau_{il}(t)]^{\alpha} \cdot [\eta_{il}]^{\beta}}$$

Where $p_{ij}^{k}(t)$ is the probability of moving from point i to point j, α and β are parameters which determine the importance of the pheromone trail and the best apparent route respectively, η_{ij} is a measure of how good the arc between i and j apparently is, $\tau_{ij}(t)$ is the pheromone trail between points i and j at time t. $\mathcal{N}_i^k$ is the feasible neighbourhood of points to which the ant can move (15). Therefore, through changing α and β, the importance of the pheromone trail and the heuristic information can be adjusted.

The process of selecting a path to take using the probability calculation above continues until the ant finishes the solution by reaching a pre-defined end criterion for the particular solution. Following this the pheromones are updated using the concepts of "pheromone evaporation" and "pheromone depositing" in the following manner:

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t)$$

Where $0 < \rho \leq 1$ is the rate at which the pheromone evaporates and m is the number of ants (number of solutions in the round). This means that far routes which the ants do not use, the pheromone will decrease, allowing the algorithm to

"remember" that these routes were undesirable. $\Delta\tau_{ij}^{k}(t)$ is the amount of pheromone which is deposited by an individual ant on a route which is chosen, and is determined in the following manner:

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \frac{1}{L^{k}(t)} & if\ arc\ ij\ is\ chosen\ by\ ant\ k \\ 0 & otherwise \end{cases}$$

$L^{k}(t)$ is the length of the ants trip. This means that if the trip is long then less pheromones are deposited thereby causing the algorithm to encourage only the most effective routes (15).

The algorithm continues in this manner until a pre-defined final termination criterion is reached, with ants not only remembering which solutions are viable with pheromones, but also which individual steps were desirable

7.6.2 Generalised Assignment Problem

One potential way in which a line balancing problem could be approached is as a Generalised Assignment Problem (GAP). The logic behind approaching a line balancing problem in this way would be to assign *n* processes to *m* work packages such that *n*>*m*. In a GAP the problem is formulated as follows and is generally explained as a knapsack problem with *n* items being assigned to *m* knapsacks with p_{ij} being the cost of assigning item *j* to knapsack *i*, w_{ij} being the weight of assigning item *j* to knapsack *i* and c_i being the capacity of knapsack *i* (20):

$$minimise \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}x_{ij} \quad (minimise\ total\ cost)$$

$$subject\ to: \sum_{j=1}^{n} w_{ij}x_{ij} \leq c_i \quad (respect\ knapsack\ capacities)$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad (all\ items\ assigned)$$

$$x_{ij}\ \epsilon\ \{0,1\}$$

The GAP is a problem which has a large amount of research covering its formulation and solution and it can be solved with some metaheuristic methods such as a Constructive Genetic Algorithm (CGA) (20). The CGA is a variation of a standard genetic algorithm. It approaches the problem by using "seed items". It assumes that certain decision variables have been correctly allocated (the "seed items") in order to simplify the solution of a particular problem. This artificially reduces the number of decision variables (20).

A line balancing problem can potentially be formulated as a GAP problem. If index *n* refers to the processes to allocate, index *m* refers to the work packages the processes are being assigned to, w_{ij} *is* the time taken up by assigning process *i* to work package *j* and c_i *is* the maximum capacity in time for work package *i,* the objective function would become:

$$maximise \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}x_{ij} \quad (maximise\ the\ number\ of\ processes\ in\ a\ package)$$

A large number of additional constraints are needed in order to prevent a process being created which does not adhere to the necessary build sequence. This is not a constraint which is normally part of a GAP, since the GAP does not differentiate the order in which the processes are allocated. This is dealt with by adding the additional constraints, or by changing the w_{ij} to discourage the algorithm from selecting an impossible build sequence.

In a line balancing problem, the number of packages *m* can vary and is not necessarily known at the start of the process. This means that the algorithm needs to be run with increasing values of *m* until a feasible solution is found. This can increase the amount of processing time by several orders of magnitude. An alternative is to add a process is equivalent to a full package for the purposes of the objective function. Otherwise a bi-objective function could be used.

7.6.3 Methods for Dealing with Sequence

Kim *et al.* (11) showed that an effective method for representing build sequence is through the use of network diagrams. These can also have additional information such as time and build area. However, the question is how to include this data in an algorithm. Some potential ways to deal with sequence are as follows:

1. **Constraining the algorithm:** This method involves adding constraints to the algorithm so that a solution which violates these constraints is considered

infeasible. Constraints, in this case, refer to rules which, if the algorithm violates them, either render a solution completely invalid or the algorithm does not even consider them in the first place. The risk of adding constraints in this manner is that any search method which is used to find a best or optimal solution may be so constrained that it is unable to find any solution. This is due to the fact that adding many constraints can create a large number of local optima which can lead to poor solutions being found. If the constraints render too many solutions in the sample space infeasible, then it could lead to no solution being found at all.

2. **Limiting the algorithm through the objective function:** Another alternative is to apply penalties to the objective function for solutions which violate the sequence constraints. The benefit of this is that the algorithm can consider solutions which are "infeasible" which means that the algorithm will not be constrained, allowing random search heuristics to be more effectively used. Care needs to be taken, however, to weight the objective function sufficiently so that an infeasible solution cannot be considered optimal.

Similarly a combination of these two approaches can be used for different constraints. The general rule should be that constraining the algorithm should be used for absolutely inviolable rules whereas objective function penalties should be used for constraint which are not as inviolable, but which simply make a solution less desirable. It is essential however to evaluate any solutions found to ensure that solutions which would be infeasible are not being found as optimal if there are feasible solutions which can be found.

8 Objectives

The objectives of the research are to evaluate the following within the scope of the frontend assembly line of BMW South Africa's Rosslyn Assembly Plant:

- Obtain details regarding the assembly process as well as the data source.
- Determine KPI's which can be used to create an algorithm for improvement of the line balancing solution.
- Develop, evaluate and test a model to provide workable solutions to line balancing problems while optimising Key Performance Indicators (KPI's).
- Evaluate the extent to which the KPI's of loading, number of work packages, stability and work area are improved.
- Show an improvement in the line balancing solution of the line being analysed.

8.1 Data Source

In order to understand and develop KPI's for the development of the model it is necessary to understand the data source and structure within BMW South Africa. The structure of the line will be described later in section 10.1. The data gathered for the purposes of testing the algorithm under development was obtained from BMW's assembly plant in Rosslyn, South Africa. The data gathered is based on BMW systems and the manner in which the company processes and performs time studies on work content in order to balance an assembly line. Therefore any model which is developed in this environment is created on the basis of using this information in its current format. The manner in which the data is stored can be summarised in Figure 7.

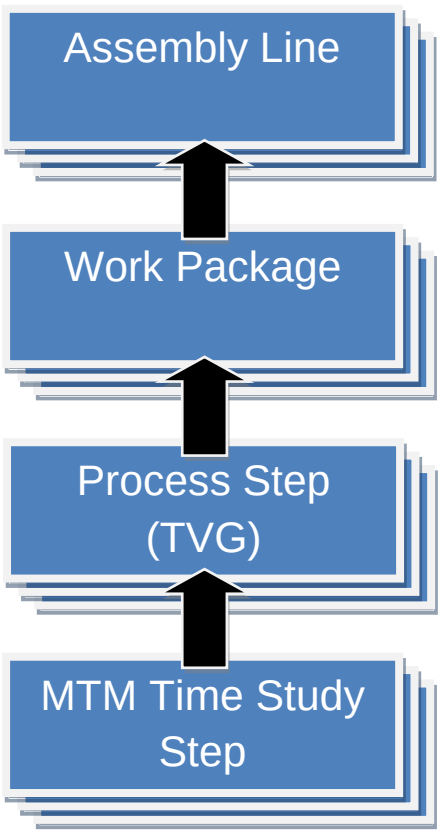


Figure 7: Overview of BMW Data Heirarchy

This figure shows how the data cascades into varying levels of detail in order to simplify the time study and line balancing processes. The lowest level is time study data in the form of Methods Time Measurement (MTM) data using company developed standard data for the time analysis. This method involves measuring standard times for common steps in a production process in a controlled environment. These measurements can then be used for time study across the entire BMW Group in order create a standardised time study process and results which are not subjective. An example of the calculation of a walking step can be seen in Table 1 where TMU are time measurement units corresponding to a pre-defined length of time.

Table 1: Example of Standard BMW Group MTM Data

Walking and Physical Movements			Code	TMU	Num. Code
walk 1 m, body > 90° step (stairs)	Turn 1	in assembly process	S-KAM	25	38 0
		get parts for car	S-KAV		39 0
		car body to car body	S-KAK		40 0
bending, stooping, kneeling incl. straightening up			S-KB	60	41 0
sitting and rising			S-KC	110	42 0

These micros steps are used to analyse production processes based on what an analyst sees during the manufacture of a vehicle or in the pre-development of the processes prior to the launch of a vehicle. The selection of all the steps which the analyst sees is then collated to analyse a process step or "TVG" (Teil Vorgang or Process Step). These process steps tend to relate to an operation which an operator performs. The scope of these is usually determined by what is deemed to be a process which cannot be split in two and which shares a common optionality. Therefore the process can be assigned common option characteristics. The fitment of a single part typically tends to be a single TVG. However, if it is decided that the process may be split across more than one associate, then the TVG would be split. Additionally if the operation varies depending on an option, then this would also be cause for a split. A series of TVG's are then put together in order to build up a work package which is a cyclical task. This is performed by an associate on the assembly line. This sequence can vary from unit to unit depending on the optional content, region, engine code and drive of the vehicle, meaning that some TVG's will only be valid for a portion of the volume.

A work package typically contains value-adding process steps, as well as the corresponding walking, parts handling and inspections which are necessary to accurately reflect the assembly process. Therefore the TVG's can have any combination of value-adding and non-value-adding steps, with many TVG's being entirely non-value adding. The sum of the times of these process steps is determined from the MTM analyses contained within the TVG's. They can be used to determine the utilisation of an associate who is performing the work package, as follows:

$$Utilisation\ (\%) = \frac{\sum PT}{TT}$$

Where $\sum PT$ is the sum of all the TVG times and TT is the takt time. This can be calculated on an average basis, or for a particular unit, if options are involved.

A TVG does not only contain information regarding the time the process takes, but also contains data regarding the options and variants which the process step is applicable to, as well as in which area the process needs to be performed. This is essential in accurately determining the utilisation of the associate on a car by car and day to day basis and to maximise efficiency through effective line balancing practices. It also reduces the walking and non-value adding time. The TVG fulfils the role of providing a centralised data source for all assembly plants building the product specified, and thereby a standard can be maintained for assembly processes between all plants.

8.2 KPI's for line balancing

When line balancing is performed on a production line there are two major key performance indicators (KPI's) which need to be considered. These measures are intended to encourage the achievement of a high level of efficiency on the production line without impacting on quality outputs from the assembly processes. The KPI's are as follows:

8.2.1 Loading

Average loading is a measure which is calculated by determining what the loading in a work package will be on average across any period of time. While this is a simple method to get an idea of the utilisation of an associate, it can be a very misleading measurement. This is due to the fact the effect of variation from product to product is missed. This can lead to line balancing being done which provides acceptable average loadings. However, when implemented, it causes dramatic effects on quality and efficiency overall, due to highly variable product loadings. This can lead to a large amount of waste as well as overloading. This effect is illustrated Figure 8. Figure 8 is an actual example of a process in the Rosslyn assembly plant. In this case the average utilisation indicates under loading and waste present in the process. However, when the minimum and maximum loadings are considered, it is clear that some work packages may be overloaded. In this case, despite appearing under loaded, 009001P is not likely to be a successful work package if additional content is added due to the variation in loading and the high maximum loading. However, 009002P can have additional content added despite having a similar average. This is due to the maximum loading in 009002P being lower than the maximum of 100%. The loading in 009002P can increase by up to 18% before overloading is experienced. In cases where the loading is close to 100%, it becomes essential for the time study info to be accurate.

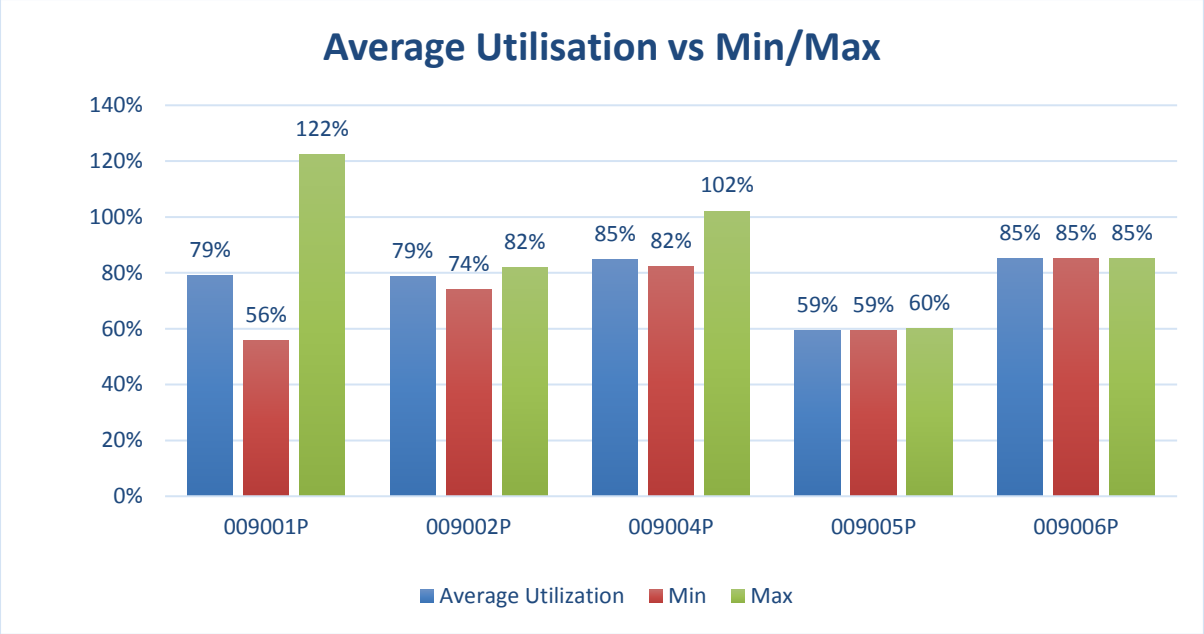


Figure 8: Example of Average vs Min/Max Loadings

8.2.2 Stability

A major concern on many production lines, and particularly in many automobile production lines, is how the system deals with the variation and customisability which many customers demand. Variations caused by optional extras, varying engine types as well as varying types of drive trains and gearboxes can have an effect on the efficiency as well as the quality output of a manufacturing process, if these complexities are not considered and planned for. The best way to manage this is through statistical measurement, and an appropriate response to outliers, in order to alleviate the effect on the assembly line. This response could be line balancing, or temporarily providing additional support to the operator experiencing the overload. One form of measurement for these complexities is called stability.

Stability is a KPI which is intended to allow for regular, measured work while avoiding an excess of waste in assembly processes. The concept of stability comes from lean philosophy of "*mura*", or unevenness, which encourages regular work in order to allow the associate in the assembly line to focus on the fitment of parts without distraction and without variations in work load. The idea behind this is that a worker who has dramatic variations in work load from product to product as well as a large amount of waste inherent in their process will not be able to obtain a rhythm, and as a result, will be distracted from producing quality work. Additionally in such a situation the associate will find it difficult to perfect the assembly process. Stability as a measure contains two key components:

- Loading across all variants of the product.

- The number of areas of the product which are worked on.

The loading component of stability is determined by considering all the possible variants of the product being assembled and identifying if any variants lie beyond the pre-defined acceptable bounds. The bounds for these, as seen in BMW's Rosslyn assembly plant, are defined in Figure 9:

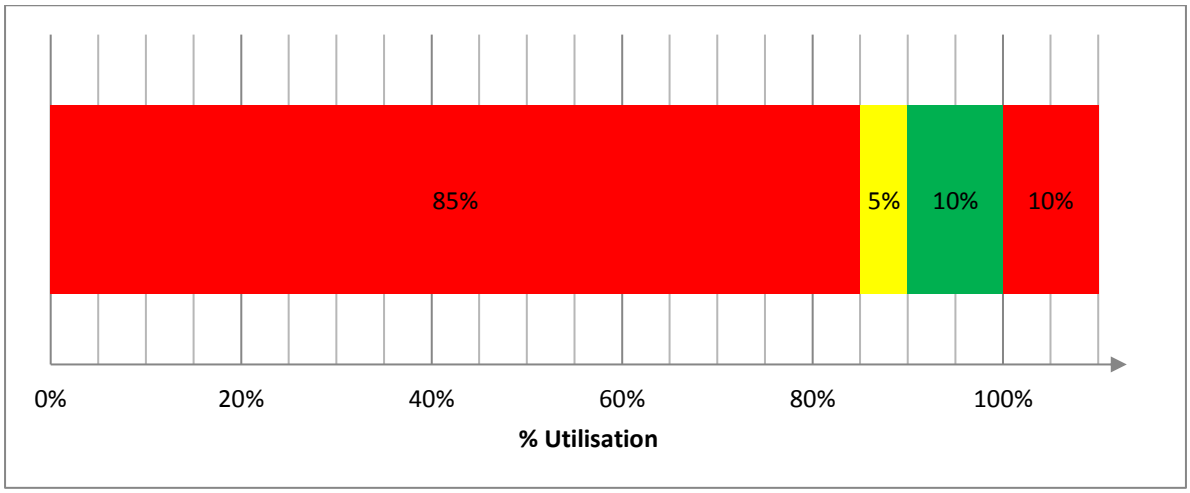


Figure 9: Stability Criteria

Any product loaded below 85% is considered unstable, anything between 85% and 90% is considered acceptable, anything between 90% and 100% is considered stable, and anything over 100% is considered as unstable. Generally the concept of stability can be applied in one of two ways. The first is that if any products are outside of the stable bounds then the work package is considered unstable. This approach is absolute, and is generally used to simplify the concept when dealing with it on a production line. All that needs to be determined is the minimum and maximum loading, and the stability of the work package can be determined. The second way is to determine the stability by percentages. This is done by calculating the percentages of the volume produced which fall in each of the stability bands. This approach provides a more accurate visualisation of the variability of loading inside a work package. However, in order to determine this accurately, software packages are required which are programmed for this purpose. An example of a stability graph which is generated from such a software package for a segment of the production line can be seen in Figure 10. This graph shows the percentages of cycles executed in the work station which fall within the loading bands. For example in 060010P 86% of the volume is above 100% load, 11% is between 90% and 100%, 2% is between 85% and 90% and 1% is less than 85%. In this graph, the packages are all stable to varying degrees. The general rule for dealing with stability is that the overload is more of a problem for an assembly process than underload. This is

particularly true if it is a high overload and/or it applies to a large proportion of the volume.

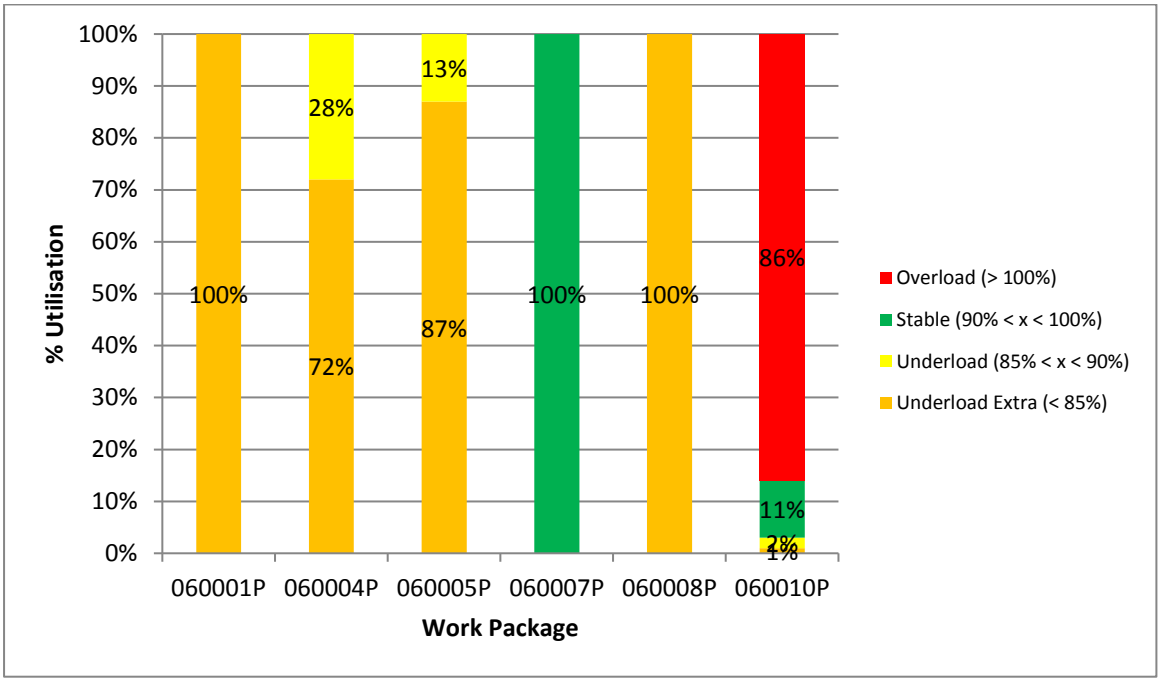


Figure 10: Example of a Stability Graph

The second dimension of stability is the work area of the product. A work package is usually defined as not being stable if the associate has to work in multiple areas of the product. This is particularly relevant because the product being assembled is large, and there can be non-value adding and wasteful steps in order to move between the different areas of the product. A package can never be stable if it is required that the associate move to multiple areas of the vehicle. Figure 11 illustrates a basic concept of work areas where the vehicle is divided into six sections labelled as *VH*, where *V* is the vertical position in the diagram and *H* is the horizontal position in the diagram. The vertical positions are defined as right (*R*), middle (*M*) and left (*L*) while the horizontal positions are defined as front (*V*), middle (*M*), and rear (*H*).

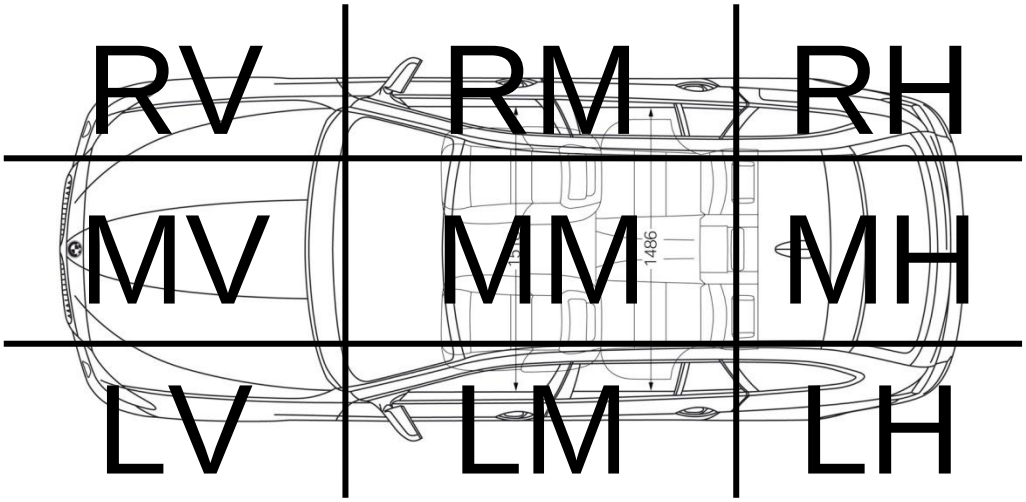


Figure 11: Diagram of Basic Work Areas on a Vehicle

Work area stability can also be measured in the form of the proportion of non-value adding steps in the work package. This requires analytical input to the process, as time studies will be required to determine the non-value adding time in the process after a line balancing has been done. This can lead to several line balancing processes being necessary.

8.3 KPI's to be achieved

Necessary KPI's:

- **No work packages within unacceptable amount of loading above 100%.** This will be measured by checking the maximum loading on each work package to confirm that the loading does not exceed this threshold. If the loading does exceed this threshold, then analysis needs to be performed in order to determine the amount of volume over 100% and how much over 100% the units are. This can be checked through the use of a Min/Max graph as shown in Figure 8 for the maximum loading, as well as through a stability graph as shown in Figure 9 to determine the amount of volume over 100%.
- **Minimising the number of work packages on the assembly line.** This is compared by checking the number of work packages which are in the final solution against the number of work packages currently on the line.

Secondary KPI's:

- **To maximise the stability on the assembly line being tested.** This is done by creating a stability graph for the line based on the stability criteria shown in Figure 9 and evaluating the stability levels from this.

- **To avoid any work packages which work in multiple work areas on a vehicle.** This will be tested by seeing the number of penalties which are incurred by the algorithm through a change in work area.

In terms of the wastes which are part of lean manufacturing methodology these KPI's offer a number of improvements. The "necessary KPI's" will maximise the efficiency through minimising the number of work packages, and without causing overloading. This ideally will target the wastes of waiting time and overproduction (*muda*) and overloading (*muri*).

The "secondary KPI's" will target criteria which have a lesser effect on efficiency but can also impact on quality output. By maximising the stability on the line, multiple wastes can be improved, such as waiting time and overproduction (*muda*), as well as the unevenness in the process (*mura*). Finally, by avoiding multiple work areas, the movement (*muda*) in the process can be reduced.

9 Methodology

9.1 Problem Formulation

A model is developed to achieve the defined improvement KPI's. The problem to be solved will vary from instance to instance as there will be different constraints which need be considered. The general form is as follows:

$$\min z = \sum_{i=0}^{N} (TT - MT_i) + \sum_{i=0}^{N} \left(\frac{\sum_{i=0}^{N} AT_i}{N} - AT_i \right) + Penalties$$

$Subject\ to$:

$x_{ab} < 1\ where\ sequence\ constraints\ apply$

$$x_{ab} = 0{,}1 \quad for \quad a = 0{,}1{,}2{,}\dots{,}N\ and\ b = 0{,}1{,}2{,}\dots{,}N$$

$MT_i < TT$

$AT_i < TT$

$$Penalties = \sum BRP + \sum EXP$$

Where *N* is the number of work packages in the solution, *TT* is the takt time, MT_i is the maximum process time of work package *i*, AT_i is the average process time of work package *i* and x_{ab} is the arc between process *a* and *b*. *BRP* is the sum of penalties for violating work areas and *EXP* is the sum of the penalties for violating constraints regarding processes which cannot share a work package.

The objective function is comprised of three parts. The first calculates the amount of waiting time across the entire solution and is based on the Line Efficiency KPI in section 7.4.2. The second calculates the variation in process time from the mean process time in order to balance the work packages and is based on the Smoothness Index in section 7.4.2.

9.2 Algorithm Decision

There are two broad types of algorithms which can be considered for the solution of a line balancing problem, these being neighbourhood search methods or constructive search methods. Neighbourhood Search methods such as Tabu Search and Genetic Algorithms search a neighbourhood for solutions. These have a built in memory function through a Tabu List and success of previous generations respectively. Constructive methods such as Ant Colony Optimisation (ACO) build up a solution step by step and have memory through storage methods which store how successful previous steps were. In the case of an ACO, this is achieved using pheromones.

The choice between a neighbourhood and constructive search algorithm depends on the problem and how it is structured as well as to what extent the search space is constrained. Generally neighbourhood search methods are applicable to many different types of problems and not as tailored to a specific problem. This is due to the fact that these methods are applied to problems that are usually not "hard-constrained", but rather the constraints are represented by penalties in the objective function (the constraints are relaxed). This means that neighbourhood search metaheuristics consider every possible solution in a sample space even if the solution violates one or more constraints. If the solution space is extremely constrained then such methods may be unable to find enough feasible solutions to make a meaningful improvement.

Constructive search algorithms such as ACO, due to the nature of their methodology, provide the opportunity to incorporate constraints into their decision making. This allows the algorithm to only search for solutions which do not violate a set of constraints and through doing this, the likelihood of finding an effective solution is increased and the time taken to do so is decreased. Constructive search algorithms, and ACO in particular, are applicable to problems where sequence is of concern. Examples include the Travelling Salesman Problem and scheduling problems. Therefore, if the algorithm is structured correctly, constructive search algorithms can be useful for the solution of highly constrained sample spaces.

Line balancing problems, due to the constraints placed by characteristics such as build sequence and work orientation, are highly constrained systems in which sequence is of high importance. Therefore a constructive algorithm like ACO should be able to be applied successfully with the build sequence, which is an inviolable constraint, being built into the structure of the algorithm's decision making.

To illustrate the reasoning behind this decision an example of a 6 task line balancing problem can be seen in Figure 13. In this problem, if a metaheuristic with relaxed constraints were used, such as a neighbourhood search algorithm, then a sample space of $6! = 720$ solutions would be considered. However there are only 6 possible

feasible solutions being feasible which do not violate any of the sequence constraints.

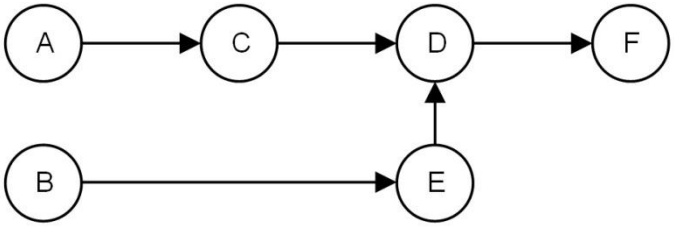


Figure 12: Example of a six task line balancing problem

A constructive algorithm which integrates the sequence constraints would only search the 6 feasible solutions and therefore would be able to find a solution more rapidly. Therefore the decision was made to focus on an ACO to solve the line balancing problem.

9.3 Inputs for Ant Colony Optimisation Algorithm Development

The problem which is used for development of the algorithm is one which is taken from BMW SA's Rosslyn plant's frontend assembly line. This is a line which provides a base from which the algorithm can be developed as it is not too large from a processing time point of view. It includes a number of different types of constraints which can be investigated and solved such as work areas, tasks to be done by multiple operators and tasks which need to be done as a group. Additionally this line is currently one which experiences a large amount of under loading and therefore provides an opportunity for a meaningful improvement to be made through the use of the algorithm.

In order to develop the algorithm a number of inputs need to be provided in a format which can be mathematically understood. These inputs are as follows:

- Build Sequence
- Process Steps
- Process Times
- Takt Time
- Work Area Information and Penalties
- Exclusivity Constraints
- Pheromone Change Rates

The approach taken in order to process the information into a readable form was, first, to obtain the list of process data for the assembly of the product on the frontend line. A sample of this data can be seen in Appendix A and shows some of the detail which can be found within each TVG. This data was then grouped into process steps where tasks have to be performed together in sequence and cannot be split. This grouping methodology can be seen in Appendix B and amounted to a total of 39 grouped process steps which could be used for balancing.

These groupings were then analysed for build sequence through interviews with associates and team members in the assembly line area and the knowledge of the author into a build sequence diagram in the same format as the one previously shown by Kim et. al (11). This visualises the build sequence and illustrates the complexity of the problem. The work areas are also visualised on this diagram for complete transparency. The complete build sequence diagram can be seen in Appendix C. The takt time input was also obtained from BMW SA for the line in question. The constraint information obtained included the work areas (Bauraum) of the line meaning the area of the frontend assembly in which it is possible to perform the process group in question as well as the exclusive processes list which refers to a list of processes which cannot be performed together in one work package. An exclusive process normally refers to a form of equipment constraint or a constraint due to two associates being required to work together on a single operation thereby necessitating the processes being split into different work packages.

These inputs then needed to be placed into a format which could be loaded into computational software for the development of the algorithm. The software used for the development of the algorithm was MATLAB™ and therefore an input of matrices was decided upon. The input of the all the data is shown in Appendix D. The sequence input data in Table 18 shows the arcs between tasks and the implication for the algorithm for choosing an arc, direct sequence arcs are shown in green, backwards (impossible sequence arcs) are shown in red and white arcs are those which indicate arcs which do not show a direct sequence relation between the two points. Additionally the work area data is integrated into this matrix in the form of the values in the matrix, a 1 indicates that the operation can be done in any work area, a 2 indicates that it must be done from the front and a 3 indicates that the operation must be done from the rear. The process which is referred to for work area data is the column in the matrix. The information for the work areas is derived from the information in Table 23 in Appendix E which shows the work area for every process in the problem.

The process step data in Table 19 simply imports the process step in a numerical matrix format which is easier to use in an algorithm, this numerical format is used in all the operations of the algorithm. The work area penalties input file which tells the algorithm what penalties are applied for selecting work area changes in a work package can be seen in Table 21 where the numbers are valued between 0 and 100

with an increasing penalty being applied. The exclusivity matrix can be seen in Table 22 and shows which processes cannot share a work package in each row.

All of the data in Appendix D: Example of Input Data for ACO Algorithm) should be placed in an external file, this file is selected via an input dialogue. The takt time and pheromone rates are input manually through an input dialogue. The takt time needs to be determined from the line being studied and is essential to the operation of the algorithm. The pheromone rates, $\Delta\tau_{ij}^{k}$ and ρ, represent the deposition rate and evaporation rate respectively. These are customisable in order to change the characteristics of the search. Recommended values are 0.03 and 0.002 respectively.

9.4 Objective Function

In any random search method the objective function determines how desirable a solution is, and is used to steer the algorithm towards finding the best solution. For this reason it is essential that the objective function truly reflects what type of solution the operation research study hopes to achieve. In the case of a line balancing problem, the objective function needs to incorporate evaluations of the KPI's such as waiting time, stability, headcount and variability. It should also contain any relaxed constraint penalties which need to be applied, as follows:

$$\min z = \sum_{i=0}^{N} (TT - MT_i) + \sum_{i=0}^{N} \left(\frac{\sum_{i=0}^{N} AT_i}{N} - AT_i \right) + Penalties$$

Where TT is the takt time, N is the number of work packages in the solution, MT_i is the maximum time in work package i and AT_i is the average time in work package i. The penalties are fixed values which are pre-determined by the algorithm. This objective function, therefore, directs the algorithm to achieve the following:

- Minimising waiting time and therefore the number of work packages through the first component of the objective function.
- Equalising the load between the work packages through the second component of the objective function.
- Discouraging the violation of any non-sequence related constraints added onto the objective function through the last component of the objective function.

9.5 Constraints

The best approach to dealing with constraints was required. It is unlikely that using hard constraints for all constraints will provide the best results, as the search for a solution may become over-constrained. This may prevent the algorithm from finding different solutions and evaluating the sample space completely. Therefore, through modification of the algorithm, the manner in which different constraints were handled was tested to try and find the most effective approach.

There are several types of constraints which can be present in a line balancing problem. They are:

- Sequence Constraints – These represent the physical order in which the product needs to be constructed. This prevents a process being chosen which is physically unachievable.
- Work Area Constraints – These represent the limitation which prevents a worker in a work package assembling parts on multiple areas of the product. This is in place to reduce the waste from moving from one part of the product to the other.
- Time Constraints – This represents the limitation to prevent the work package exceeding the takt time. This is necessary to avoid a process which is unachievable within the takt time and which would therefore either stop the assembly line or require additional resources.
- Exclusivity Constraints – This represents the constraint on any processes which are unable to be placed in the same work package. This could be due to operators working together on a single operation, structural and space constraints or due to quality concerns which may arise from the two processes being in one work package.

The application of these constraints varies in the algorithm depending on the consequences of violating them. The sequence constraints are included as hard constraints in order to avoid their violation entirely. There are two reasons for this: the first is to avoid a process which is impossible and the second is to reduce the search space for the algorithm. Similarly, the time constraint per work package is hard constrained so that it is never violated. This is also done to avoid an impossible process being generated.

Other constraints which were included as soft constraints were considered to be either too infrequent to form part of the structure of the algorithm, or would not make the process impossible, but rather undesirable. These were therefore included in the objective function as penalties, $Penalties = \sum_1^C P_C$. An example of a constraint of this type is the work area constraint, where if a work package requires that the associate works in multiple areas of the vehicle, then an appropriate penalty gets applied to the objective function.

9.6 Pheromone Rules

The approach to pheromone adjustment in this algorithm is to evaporate the pheromones by a fixed amount and then add pheromones based on the solution and the feasibility of the solution when compared to previous solutions. The reasoning behind this is that as the algorithm finds better solutions the effect of worse solutions becomes progressively less. This is done using the calculation:

$$\tau_{ij}^{k+1} = \left(\tau_{ij}^{k} \times \rho\right) + \left(\left(\frac{OF_{Best}}{OF}\right) \times \Delta\tau_{ij}^{k}\right)$$

Whereτ_{ij}^{k} is the pheromone level at iteration k between points i and j, OF_{Best} is the best objective function value obtained thus far, OF is the current objective function value and $\Delta\tau_{ij}^{k}$ is the pheromone deposition rate. This is a scalable calculation which can work for any line balancing problem. Its effect will depend on the type of results which the algorithm obtains. The pheromone levels in all the arcs between points are set to 1 at the start, and are adjusted with the above formula at every iteration.

The evaporation of the pheromone levels is done at every iteration and works by reducing the pheromones by the evaporation constant which is, by default, set to 0.002. This means that on every iteration, all pheromones will reduce to 99.8% of their previous value. Pheromones with larger values will evaporate faster than lower values, and lower values will never reach a zero pheromone level.

9.7 Algorithm Logic and Construction

9.7.1 Input Data

The logic flow of the algorithm created can be seen in Appendix F. The algorithm begins by requesting the user to input the following information which can be seen in the steps before the main loop in the flow chart in Appendix F:

- **Sequence data**: The sequence data matrix is a representation of all the arcs in the problem being considered. This data is set up in the form of a square matrix and is input in order for the algorithm to determine and take into account the build sequence of the assembly process in question. This file is the core data source of the algorithm as it is what allows the algorithm to rapidly achieve an effective solution. The sequence data is input in a matrix format and works by reading the y-axis as the previous point, and the x-axis

being the point which the algorithm is going to. By this logic, the matrix represents the full selection of arcs within the sample space. The way in which this matrix restricts the sample space is by using negative values to indicate when a point cannot be used. A negative value under a row in the matrix indicates that if that point were to be selected then sequence would be violated. The adjustment of this matrix in subsequent iterations allows the algorithm to adapt its selection process to the current situation. The absolute value in this matrix also allows the algorithm to read what work area the process in the column falls under.

- **Data point list:** This list is a single column matrix indicating the number of processes in the list and is used to control how many times loops in the algorithm run.
- **TVG times:** This is another single column matrix of the same size as the data point list which is used to determine the length of time which a process step takes. This is used to split the solution into work packages after the algorithm has found an acceptable solution.
- **Takt time:** This is an input of the takt time which the user wishes to base the line balancing on. This is information which the user needs to obtain from the manufacturing environment/assembly line management. It determines how much work content can be added to a certain work package.
- **Pheromone rates:** This is an input of the pheromone deposit rate as well as the evaporation rate. These values determine how fast the pheromones are accumulated and removed from the pheromone table respectively.
- **Work area penalties:** This matrix represents the penalties applied for moving between two work areas in one package.
- **Exclusive process list:** This matrix represents processes which cannot be placed in a single work package together.

These inputs are done through dialogue boxes which are built into the software package (MATLAB™). The sequence data, data point, TVG time, work area and exclusive process inputs are collated in a spreadsheet while the takt time and pheromone rates are entered manually. The default takt time is the time for the frontend assembly line, while the pheromone rates are those which provided good results in initial testing.

The algorithm, following these inputs, then creates an initial pheromone table (a square matrix of ones the same size as the Sequence Data matrix), sets a starting best objective function, and stores all the starting information.

9.7.2 Primary and Solution Generation Loops

The primary and solution generation loops are shown in Appendix F as all the steps contained in the grouped area.

The primary loop of the algorithm is responsible for controlling the stopping criteria for the algorithm. The stopping criterion chosen is to stop if no improvement is achieved in 1000 iterations.

At the start of the primary loop the starting process (point) is chosen. This is done by looking for rows in the sequence data matrix which do not contain any negative values. These rows represent points which, if selected, will not violate a sequence constraint. It is through this selection mechanism that it can be said that the algorithm is “hard constrained”, as it is impossible for a step which violates the assembly sequence to be selected. The selection of the starting point is done through random selection, to ensure the algorithm tries multiple, different starting points. No pheromone is used, since then certain starting points would begin to dominate the algorithm’s selection process.

The next step is to relieve sequence constraints which, as a result of the starting point selection, would no longer lead to a sequence violation. This is done by considering the values in the column of the starting point which has been selected, and reducing any negative valued arcs in the sequence data matrix to zero so that they no longer constrain any point. This will allow future selection processes to find these points as they may no longer contain a negative value meaning that all proceeding steps have been selected.

After the selection of the starting point the second loop in the algorithm begins, with the goal of generating the remainder of the solution string one point at a time. Therefore this loop runs only as long as the length of the data point list matrix. The first step in this loop is to determine what points can feasibly be chosen without violating the constraints in the current sequence data matrix, and without considering points which have been chosen in this string. This is done by again looking for rows in the sequence data matrix which contain no negative values. However this time the algorithm will ignore any row number which is found in the current solution string so as to avoid process duplication.

Once all the feasible next steps have been determined and placed in one matrix a Monte Carlo selection process is used to determine which step is selected next. This selection process is based on the current pheromone value in the pheromone matrix corresponding to the arc between the previous point in the solution string and the point being considered. The size of the pheromone value determines the likelihood of the Monte Carlo selection choosing the arc in question. This is done directly using the pheromone matrix values and these values are not manipulated or weighted in any way prior to this selection process. A random number selection process is used as per standard Monte Carlo selection methods.

Following this selection, the Sequence Data Matrix is updated by removing negative values in the column corresponding to the data point which has just been selected. This will free up points which will no longer violate sequence constraints for selection

in subsequent steps. Following this step the smaller loop for this string in particular begins again to find the next step in the solution string and repeats until the full string is complete.

9.7.3 Work Package Splitting

The work package splitting process can be seen in Appendix F as the second step after the solution generation loop ends.

On solution string completion, a feasible solution should be obtained which does not violate any sequence constraint. This solution is represented as a string showing the sequence of activities to be completed in the assembly area. At this point this string represents the sequence for the entire line and not particular work packages. Therefore this needs to be split into work packages in order to provide a usable output for the assembly process. This is done by adding process steps to the current work package (starting at 1) until the sum of the time data for the steps chosen (taken from the TVG Time Matrix) reaches the point where adding another step will exceed the takt time. At this point the algorithm moves to the next work package. This process continues until all steps from the solution string have been added to a work package. The algorithm can be adjusted to use maximum or average times as required (maximum ignores options and uses process time directly). An important characteristic of this work package cutting process is that the work area penalties are added here. These are integrated in two ways: the first is that if the work package cutting process includes work areas which would experience a penalty then it is here that the algorithm would calculate the sum of work area penalties. The second is that functionality is included here to give the algorithm the opportunity to move to a new package instead of violating the work area. This is done by calculating a proportion which determines the likelihood of the work package finishing. If the penalty is 100 (the maximum), then the algorithm will always skip to the next work package. However, if the penalty is 20 then there will be a 20% chance of the algorithm skipping the package. This process widens the feasible solution space and accounts for the fact that in many cases it may be preferred not to move between work areas but rather to move the process to a new work package. Throughout this process, the total time of each work package is also calculated for the purpose of determining the objective function.

9.7.4 Objective Function and Pheromone Updates

The final step in the main loop is the updating of the objective function and pheromones. This can be seen in Appendix F at the end of the flow diagram.

Following the allocation of tasks to work packages, the calculation of the objective function takes place. This takes place through three calculations, the difference of all the work packages from the mean work package time summed, the difference of the work packages from the takt time summed and the sum of penalties. In this step the exclusive process constraints are checked for violations by checking if the processes relevant to the constraint can be found together in one work package. If this is the case a penalty equalling the takt time is added. The penalty is equal to the takt time as the violation of an exclusive penalty would require the addition of an additional work package. The exclusive penalties are added to the work area penalties calculated in the work package splitting process to determine the total penalties for the solution. The three values are then added to determine the objective function value.

The objective function is compared against the previous best solution in order to determine if an improvement has been made. If there has been an improvement then the new best solution's objective function and work package structure are recorded for later reference. Then the solution is recorded to a reference matrix, even if the solution is not an improvement.

The final step in the main loop is to calculate the pheromone updates. This is done by first evaporating the pheromone values according to the evaporation rate across the whole pheromone table as previously discussed. Then pheromones are deposited in arcs which were used in the solution. This is done by taking each arc in turn and depositing pheromones based on how effective the solution found was by comparing the current solution to the current best solution using the ratio $\frac{z_{Best}}{z_{Current}}$ where *z* is an objective function result. The loop then repeats, if the stopping criterion has not been reached.

10 Results and Discussion

10.1 Line Structure and Equipment

BMW South Africa's Rosslyn Assembly Plant operates on a takt time of 4.13 min and produces the BMW 3-series (F30). The line is a single model line which, for the majority of its length, runs in a straight line. Therefore this assumption is made. The process layout in the assembly plant varies by assembly area with some areas running as a single sided assembly line, and others running as two sided assembly lines. The entire plant runs as a series of Paced Synchronous Production Lines (3) connected together with each line running separately and connected to the other by a series of buffers. This layout can be seen in Figure 14.

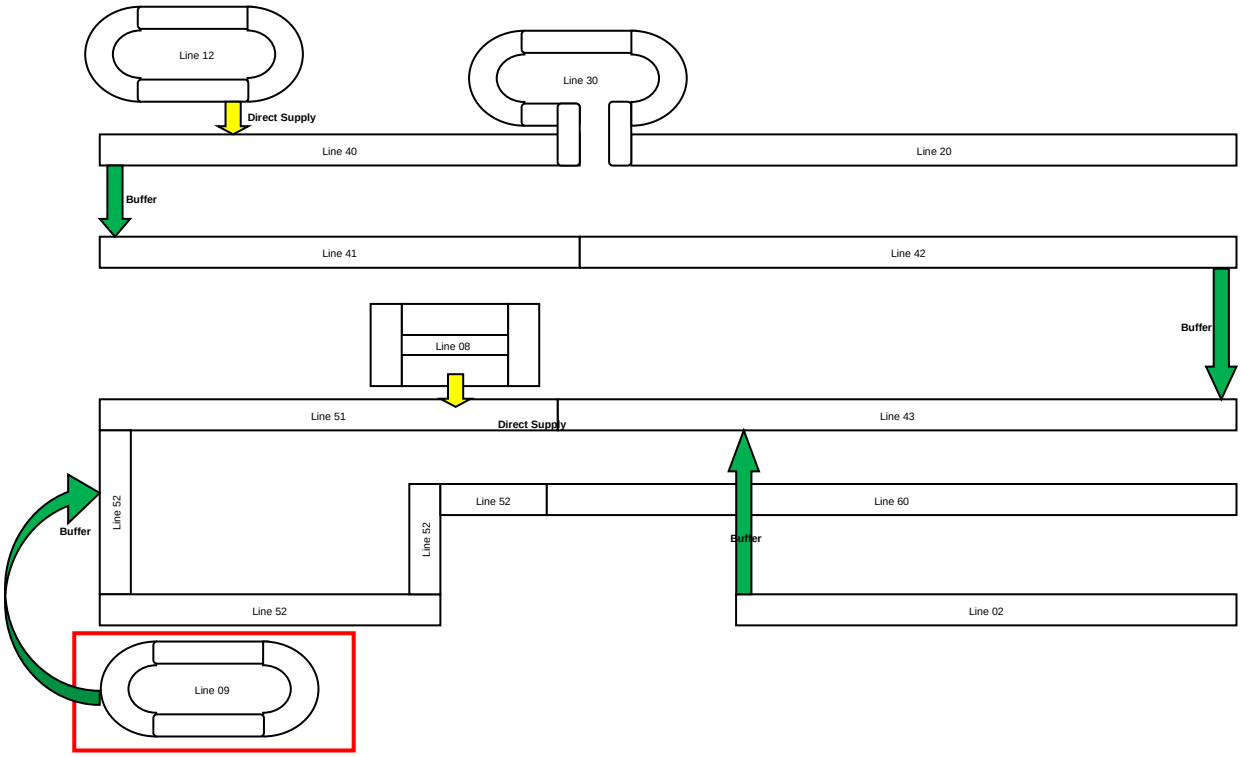
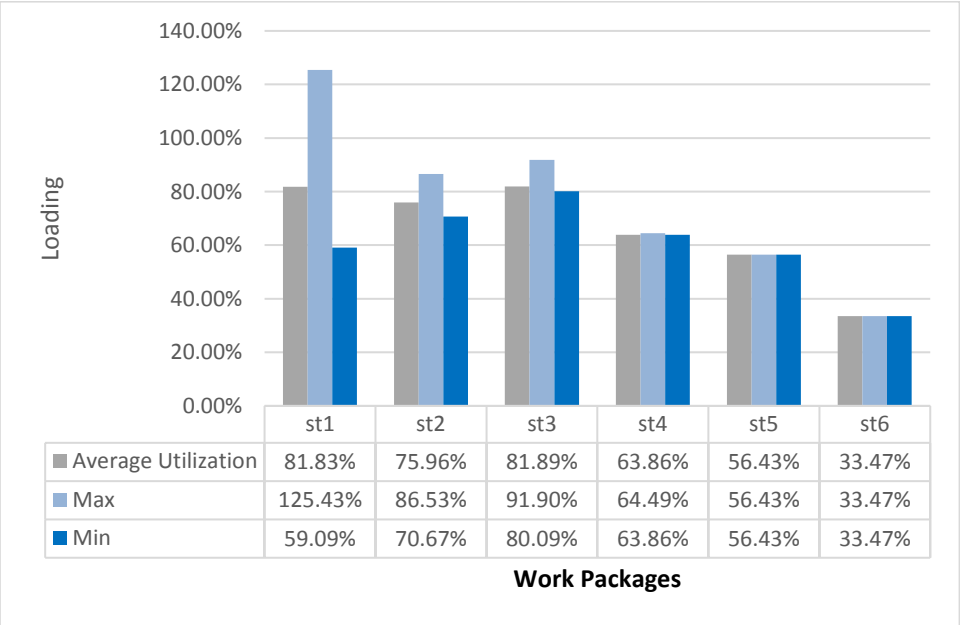


Figure 13: Plant Rosslyn Assembly Plant Layout

The current structure of the frontend assembly area (line 09) consists of six work packages containing the entire sub-assembly. Following the sub-assembly the frontend is fitted to the body of the vehicle on the main assembly line. However the transport of the frontend sub-assembly to the main line is done through a buffer system, and therefore the frontend line is considered as a separate entity.

10.2 Current State

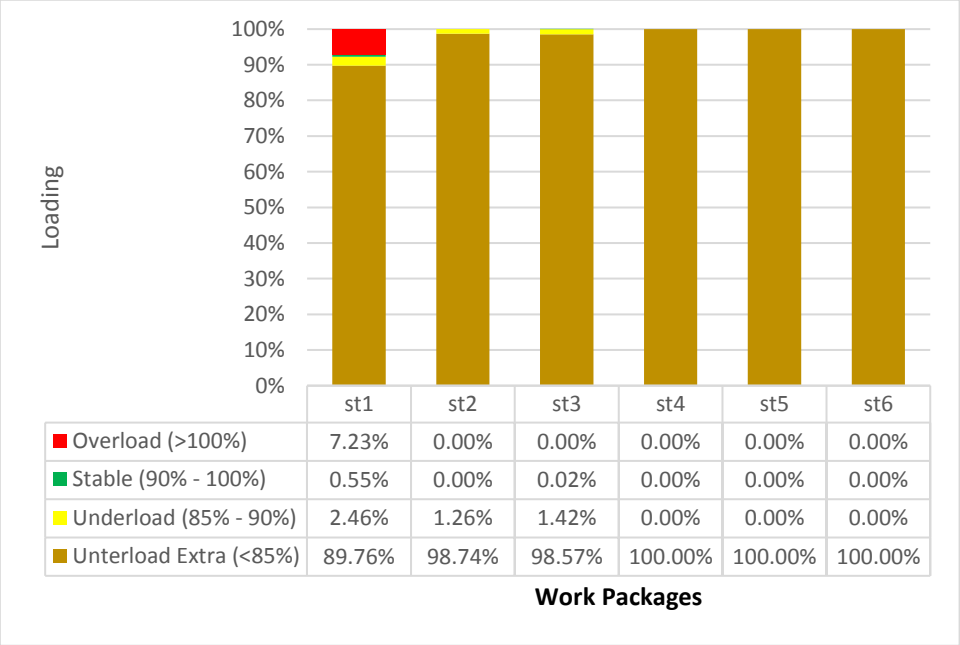
The loading graph for the frontend assembly line can be seen in Figure 14. The average loading is lower than 100% and the maximum loading is highly variable with the first work package having a maximum loading of 125% which greatly exceeds the maximum acceptable loading. The planned volume which exceeds 100% loading amounts to 7.23% of the total volume as can be seen in Figure 15, indicating that there is a risk of working too long. Support in the form of a relief worker may be needed to keep up and, in bad cases, the line may stop. A loading of 125% is a large overload which, in the case of two back-to-back units, could cause the associate to stop the line for more than a minute. This risk is combined with the fact that back-to-back units may not be an infrequent occurrence, as 7.23% is a significant portion of the volume.



	st1	st2	st3	st4	st5	st6
Average Utilization	81.83%	75.96%	81.89%	63.86%	56.43%	33.47%
Max	125.43%	86.53%	91.90%	64.49%	56.43%	33.47%
Min	59.09%	70.67%	80.09%	63.86%	56.43%	33.47%

Figure 14: Min/Max Graph for Current Frontend Assembly

Despite this overloading, it can be seen that the average and minimum loadings for this line are well below target. The ideal average loading should be in the region of 85 – 100%, and it is clear from Figure 14 that the average loadings do not reach this target. In particular stations 4, 5 and 6 are under loaded and therefore some optimisation is needed in order to reach the target loading. This under loading is apparent in the stability graph in Figure 15, where it can be seen that the vast majority of the volume is loaded at less than 85% across all the work packages, with 97.85% of all the cycles worked across all the work packages being loaded within this category.



	st1	st2	st3	st4	st5	st6
Overload (>100%)	7.23%	0.00%	0.00%	0.00%	0.00%	0.00%
Stable (90% - 100%)	0.55%	0.00%	0.02%	0.00%	0.00%	0.00%
Underload (85% - 90%)	2.46%	1.26%	1.42%	0.00%	0.00%	0.00%
Unterload Extra (<85%)	89.76%	98.74%	98.57%	100.00%	100.00%	100.00%

Figure 15: Stability Graph for Current Frontend Assembly

The average metrics for the area can be seen below:

Average Line Loading: 65.57%

Average Max Loading: 76.38%

Average Min Loading: 60.60%

Number of Work Packages: 6

Therefore, based on the current state, there is improvement which can be made. The overloading in station 1, on a small amount of the volume, is a relatively minor issue when considering the under loading across the line. Therefore, while this overloading needs to be addressed, the focus should be on increasing the loading

across the entire line in order to shift more of the volume into the 85% – 100% loading band.

10.3 Algorithm Validation

An important step before obtaining results from the algorithm is its validation. This is necessary in order to determine whether the model can be relied on to provide results which reflect a real world situation. If a validation process is not followed then the model cannot be relied upon as a source of data due to the possibility of bugs in the model and therefore errors in the data obtained from the model.

The verification approach taken for the line balancing model is to confirm key aspects of the algorithm based on solutions obtained. The steps which are taken are process and sequence validation, time and objective function validation and penalty validation.

10.3.1 Process Validation

In order to validate the process and sequence adherence of the algorithm, three solutions were obtained using the algorithm, and these solutions were tested using the sequence chart developed seen in Appendix C. The three solutions which were obtained are shown in Table 2, Table 3 and Table 4, with the work packages shown as columns, and process steps indicated as numbers corresponding to their order.

Table 2: Validation Test Run 1 Solution

WP1	WP2	WP3	WP4	WP5	WP6
4	6	16	14	37	27
1	3	15	22	17	38
7	5	12	32	13	28
11	8	9	33	30	36
21	31	23	34	18	26
2	10	20	35	25	39
0	19	0	0	0	24
0	0	0	0	0	29

Table 3: Validation Test Run 2 Solution

WP1	WP2	WP3	WP4	WP5	WP6
11	17	20	18	33	25
21	6	15	31	34	26
1	3	10	14	30	27
7	5	12	37	35	38
4	8	19	13	32	36
2	0	16	9	0	28
0	0	0	23	0	39
0	0	0	22	0	24
0	0	0	0	0	29

Table 4: Validation Test Run 3 Solution

WP1	WP2	WP3	WP4	WP5	WP6
11	6	14	16	23	39
7	3	20	22	13	24
21	15	12	32	30	29
4	5	10	33	35	0
1	8	37	34	31	0
2	0	9	0	17	0
0	0	19	0	18	0
0	0	0	0	25	0
0	0	0	0	36	0
0	0	0	0	27	0
0	0	0	0	28	0
0	0	0	0	38	0
0	0	0	0	26	0

The solutions are validated by comparing step by step to the master sequence diagram in Appendix C, in order to confirm that no sequence constraints are violated. When the solutions in the tables above were compared to the sequence diagram, no violations of the build sequence were found. Additionally, the solutions above satisfy the requirement for non-repeatability in a solution, as there were no repetitions found. Therefore it can be said that the algorithm successfully constructs solutions which do not violate sequence or non–repeatability constraints.

10.3.2 Time and Objective Function Validation

The second part of the validation process requires the confirmation of the work package times and objective function. This is key as these two criteria determine what the model considers as an effective solution. For this purpose the same test solutions generated for process validation (Table 2, 3 and 4) will be considered.

The test solutions used a takt time of 232s and included both work area and exclusivity penalties. The work area penalties were set to a penalty of 100, which is the threshold, above which none should appear in the algorithm. Table 5 and Table 6 show the objective functions/ penalties and the work package times respectively.

Table 5: Validation Objective Functions

Runs	Best OF	Penalties
1	297.2229	0
2	468.3343	232
3	366.1789	0

Table 6: Validation Work Package Times

	WP 1	WP 2	WP 3	WP 4	WP 5	WP 6
Test Run 1	226.5	213.12	221.52	182.34	204.3	146.58
Test Run 2	226.5	228.06	228.54	190.08	152.64	168.54
Test Run 3	226.5	221.4	231.72	196.26	230.34	88.14

In order to confirm whether these values are correct a manual calculation was performed. It was found that the calculations of both the objective functions as well as the work package times in all three algorithms were correct.

10.3.3 Penalty Validation

In the initial algorithm testing phase it was found that the constraints used for sequence had a conflict with the exclusivity constraints. The constraints on process steps X and Y (24 and 25), as well as AC and AD (29 and 30), are subject to exclusivity constraints. This is as a result of the processes requiring parallel work by different associates, with one associate performing one operation and the other associate performing the other operation. Due to their close proximity in the

sequence of the operation, the algorithm was unable to find a solution which did not include the two solutions in a single work package. Therefore the solution to this was to manually change the sequence matrix in order to allow one of the operations in each of the exclusive process pairs to be performed earlier in the sequence. This simulates the process of an associate in a previous workstation walking to assist the other associate with the task. This manual change can be seen in Appendix C as the first modification to the sequence constraints.

The final aspect which needs to be tested is whether the penalties in the algorithms are applied correctly once the changes have been made. As can be seen in Table 5 only the second run encountered a penalty in its best solution. In this case the penalty is confirmed in the solution as the exclusivity constraint penalty on process steps X and Y (24 and 25) is violated. The penalty applied is equal to the takt time of 232s. The work area penalty is also applied correctly, as the highest penalty is applicable for moving from the front to the rear of the work piece carrier or vice versa. A violation of these constraints must be impossible. In other runs during the development of the bauraum constraints in the model the penalty was found to be running correctly, even if only a small penalty is applied.

10.4 Algorithm Results

In the process of obtaining test results the approach was as follows:

1. Determine input criteria for model.
2. Run model with input criteria ten times.
3. Determine best solution from all iterations.
4. Evaluate solution using line balancing analysis tool.
5. Determine changes in input criteria for next model iteration.

In this way, the ideal criteria for achieving a line balancing which is efficient can be found.

10.4.1 Test Group 1

The settings to use for the first test data set needed to be determined. The takt time input into the model was the actual takt time which can be observed in the production line in its current state. This is the obvious starting point for the algorithm as it should, at least initially, reflect the line's true state as closely as possible. The pheromone deposit and evaporation rates were chosen as the values which were found in the algorithm testing and validation phase to provide the best rate of

convergence on a solution, and without restricting the ability of the algorithm to search large areas of the feasible region.

The first test run using the model was done with the following inputs:

Takt Time: 232s

Pheromone deposit rate: 3%

Pheromone evaporation rate: 0.2%

The algorithm was run ten times and then a best result was drawn from each of these test runs. These best results are shown in Table 7.

Table 7: Best Solutions from Test Group 1

Runs	Best Obj	Penalties
1	297.2229	0
2	468.3343	232
3	366.1789	0
4	469.8218	232
5	378.4176	0
6	338.9464	0
7	363.3165	0
8	378.4618	0
9	334.3084	0
10	338.9464	0

From Table 7, the algorithm found solutions with no penalty violations 80% of the time. Only one solution was found which had a much lower objective function value than the other solutions (Run 1).

Figure 16 illustrates the current best solution for the test runs over time. All the test runs are plotted on one graph to visualise the performance consistency of the algorithm.

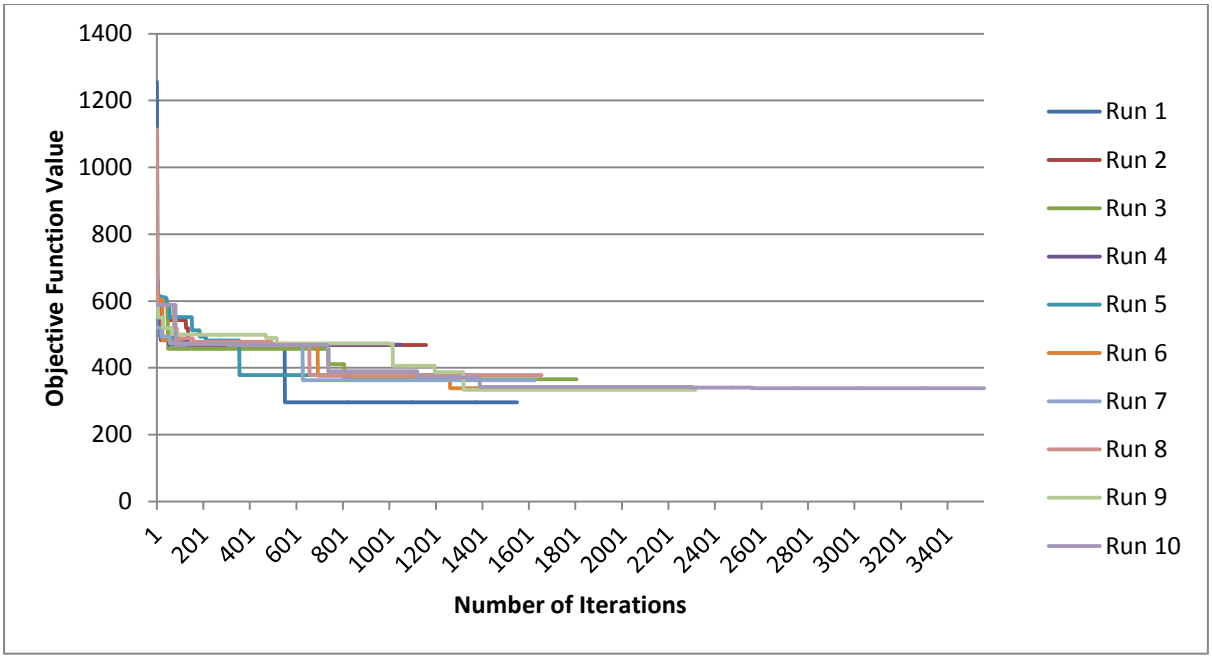


Figure 16: Test Group 1 Convergence Graph

This graph shows how effectively the model improves the solutions for all runs. The runs, with the exception of run 2 and run 4, show steadily improving solutions, and at similar rates. Certain runs had a best solution which included a penalty. Therefore these best solutions had undesirable solutions. The manner in which the majority of these runs converge on similar best solutions suggests that the algorithm is converging on an effective solution in an efficient manner.

An important part of the solution is the loading data. This is obtained from line balancing software, which compares the line balancing solution to the order data of units built in previous months in order to obtain the minimum, maximum, average and stability data. The loading data is represented as a graph showing the average, minimum and maximum loadings in a station for all the work packages in the solution. For the best solution for Test Group 1, the average metrics are as follows:

Average Line Loading: 65.57%

Average Max Loading: 76.38%

Average Min Loading: 60.60%

Number of Work Packages: 6

Clearly there has been no change in the average metrics from the current state. The reason for this is that there has been no change in the number of work packages on the line. Therefore, no matter where the loading is, the averages will remain the same. The specific values from each work package in these loading metrics can be seen in Figure 17.

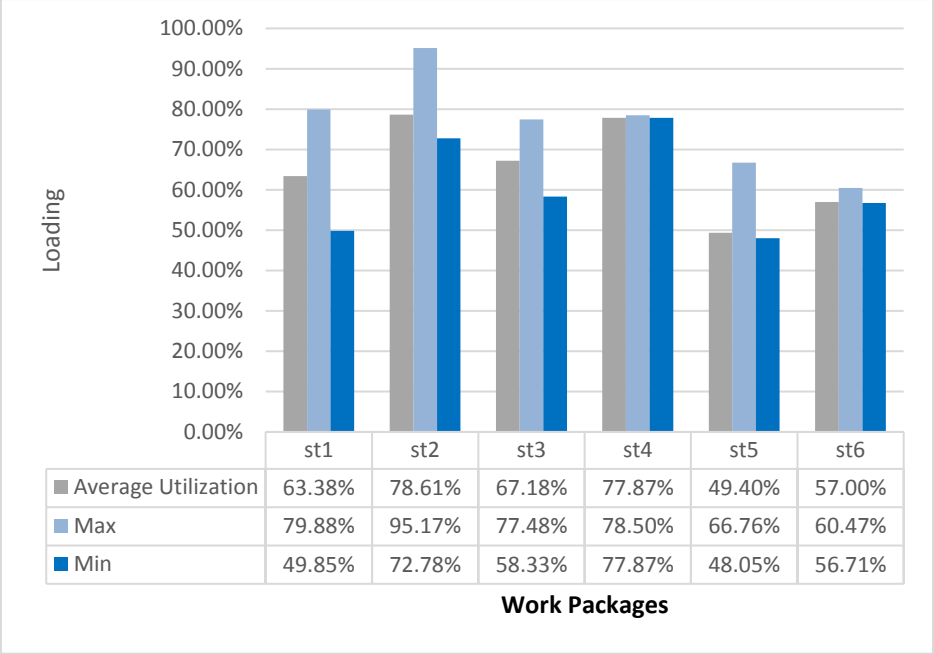


Figure 17: Test Group 1 Best Solution Min/Max Graph

Figure 17 shows a more even distribution of work load than in the current state, with the overloading condition which was seen in the current solution in Figure 14 being eliminated. This is an improvement from the current state, even though the number of work packages and average metrics have not improved

In order to better understand the results from the test group, the stability data from the best solution in the test group is examined and shown in Figure 18.

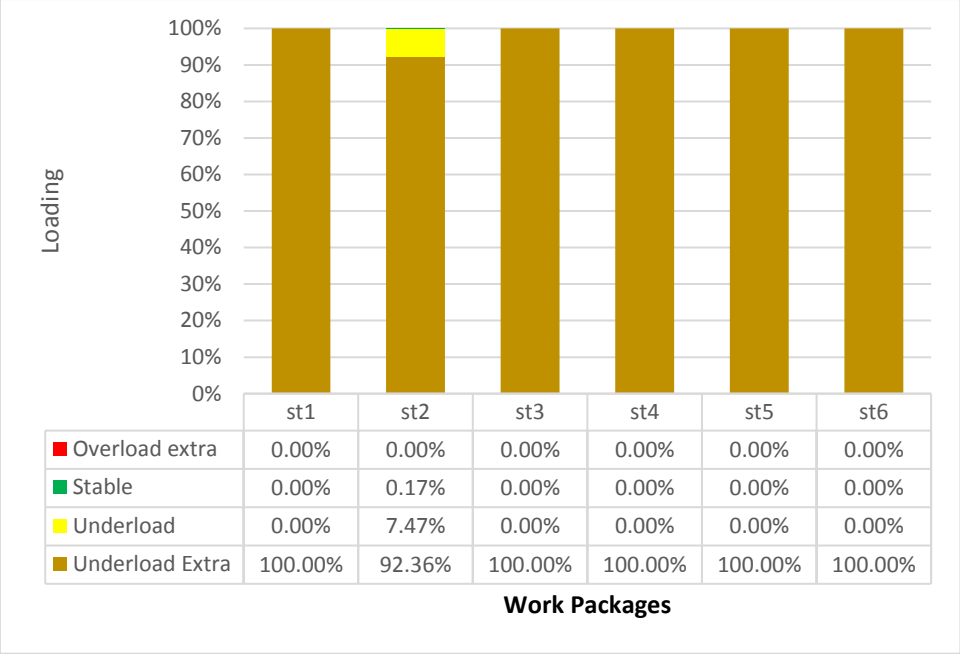


Figure 18: Test Group 1 Best Solution Stability Graph

As the convergence data in test group 1 indicates, the model is successfully finding better solutions and is finding effective best solutions. When the loading data and stability charts are considered, it can be seen that the average, maximum and minimum loading groups are all significantly lower than the takt time with the average maximum loading being at 76.38% and the average loading for the line being 65.57%. The objective was to maximise these numbers in order to increase the efficiency of the line for the next test group. Therefore the takt time was increased in order to encourage the model to add additional content to the work packages.

10.4.2 Test Group 2

For the second test group, the takt time is increased by ten seconds.

Therefore the second test group was done with the following inputs:

Takt Time: 242s

Pheromone deposit rate: 3%

Pheromone evaporation rate: 0.2%

The best results from all ten test runs is seen in Table 8. All the solutions had no penalties, and all solutions have similar objective function values.

Table 8: Best Solutions from Test Group 2

Runs	Best Obj	Penalties
1	374.4611	0
2	433.6583	0
3	399.6475	0
4	385.8523	0
5	386.7494	0
6	360.4583	0
7	348.7723	0
8	390.0006	0
9	391.7464	0
10	373.5607	0

In the convergence graph for the second test group (Figure 19), it is clear that all runs are converging on similar solutions at similar rates, as was found in test group 1. However, in this case there were no exceptions, and all runs showed a good rate of convergence.

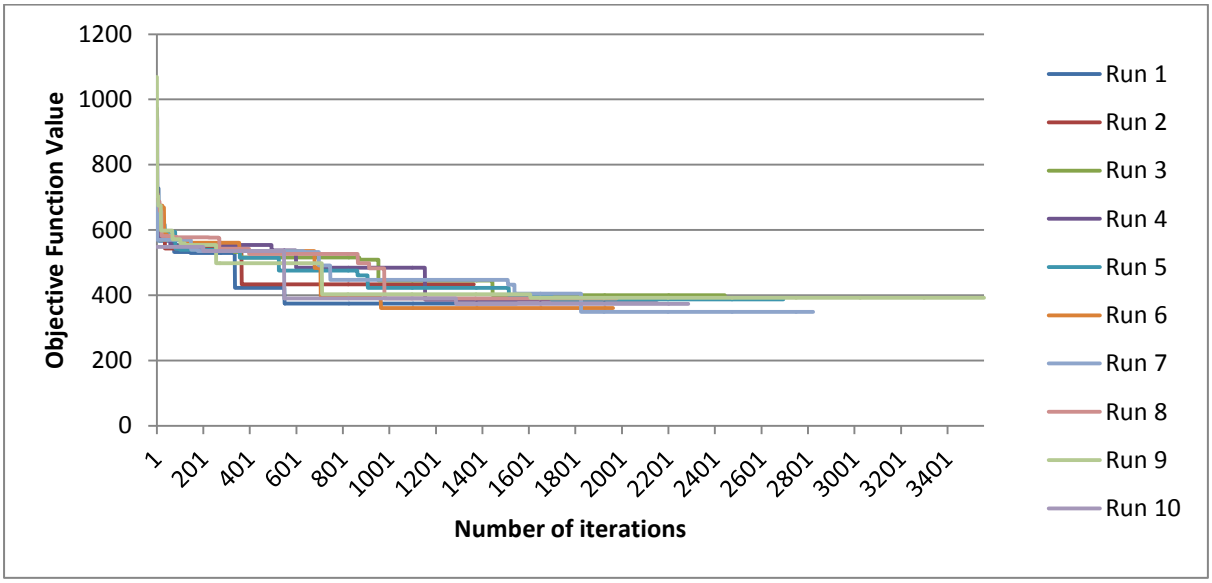


Figure 19: Test Group 2 Convergence Graph

The average loading data for test group 2 is almost identical to test group 1 and to the current state. However, it can be seen that the loading variance between packages is much less, indicating a more equal line balancing. The average loadings will not change significantly if the number of work packages does not change, since the amount of work content is constant.

Average Line Loading: 65.57%

Average Max Loading: 76.71%

Average Min Loading: 60.60%

Number of Work Packages: 6

The lower loading variance can be seen in Figure 20, and could be due to lower constraints on the algorithm due to the higher takt time. However, despite this increase in the takt time, it can still be seen that there is major under loading in many of the packages, most notably in station 5 which is loaded, on average, at 54.62%. Therefore, on the next iteration, a change was needed in order to increase the average loading on the line.

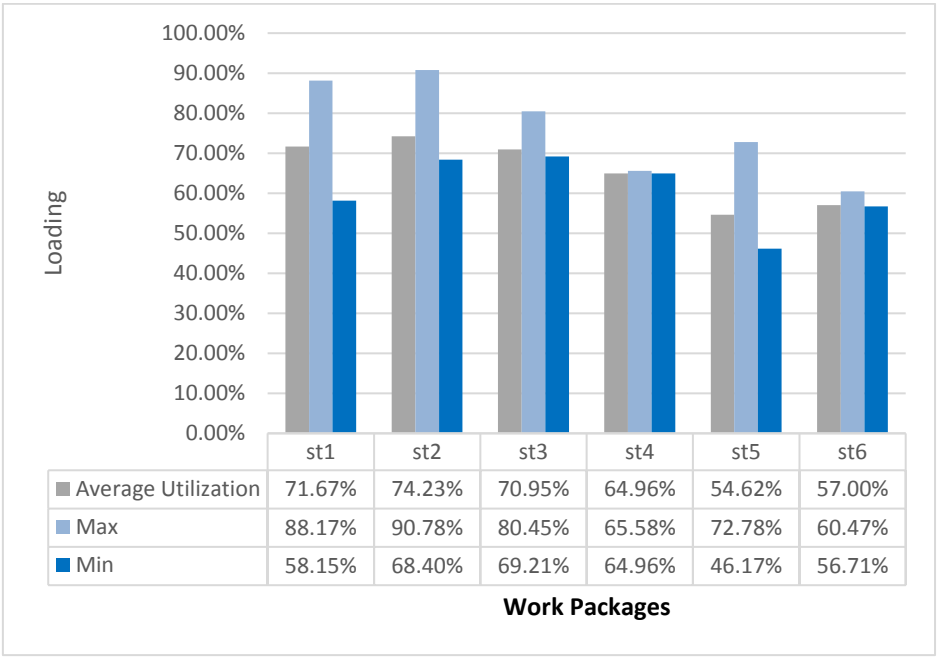


Figure 20: Test Run 2 Best Solution Min/Max Graph

The stability graph in Figure 21 shows that every package is severely under loaded as nearly all the cycles performed were in the lowest stability band (< 85%).

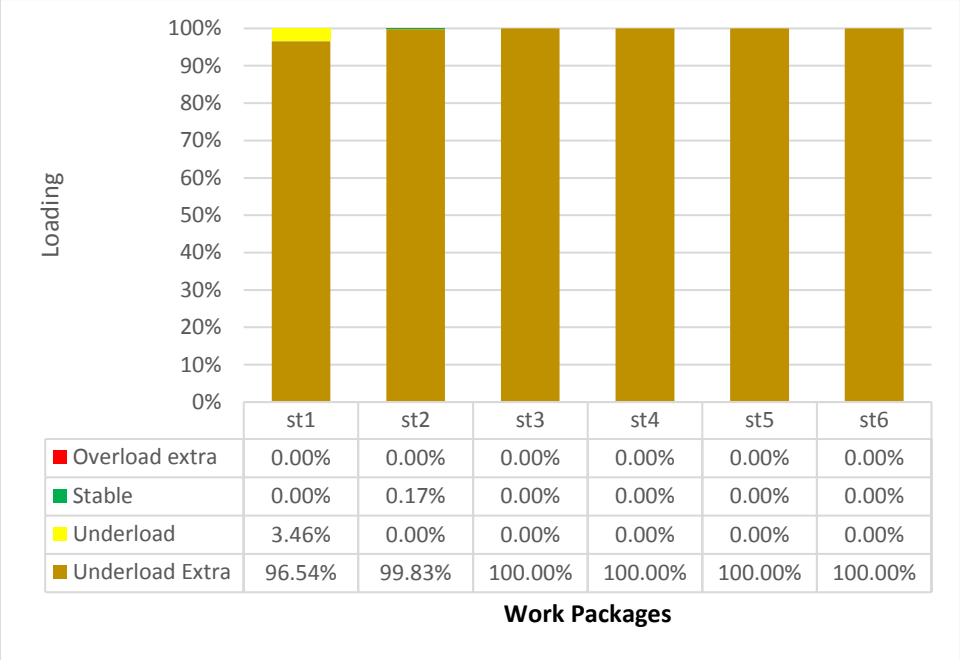


Figure 21: Test Group 2 Best Solution Stability Graph

Based on the under loading shown in Figure 20 and Figure 21 it is clear that a similar action has to be taken to that which was used after test group 1. The convergence graph indicates that the pheromone rates do not need to change, and under loading is still present. Therefore the takt time was increased further.

10.4.3 Test Group 3

For test group 3 the takt time was increased by ten seconds, in order to encourage the algorithm to achieve higher loadings.

The third test group was done with the following inputs:

Takt Time: 252s

Pheromone deposit rate: 3%

Pheromone evaporation rate: 0.2%

The best solution results from the ten runs in this test group are shown in Table 9.

Table 9: Best Solutions from Test Group 3

Runs	Best Obj	Penalties
1	440.1206	252
2	395.3551	252
3	360.3966	252
4	373.2619	252
5	417.5304	252
6	398.8923	252
7	402.6225	252
8	465.991	252
9	372.2189	252
10	373.1051	252

Table 9 shows good results but there was a penalty applied on every best solution found. When the detailed solutions were analysed it became apparent that the exclusivity constraint on steps X and Y was being violated, despite the modifications to prevent this (section 10.3.3). The reason for this is that the sequence constraints force the process to be done at a later stage. This does not take in to account the nature of this process, which requires two associates to work together. The takt time increase allows more processes to be placed in a package, which forces the algorithm to violate the sequence constraints. A sample solution from this test group can be seen in Table 10, and the processes incurring the penalty are highlighted. It is clear that the algorithm cannot provide a solution which splits these two processes.

Table 10: Sample Solution from Test Group 3

Station 1	Station 2	Station 3	Station 4	Station 5
G	O	S	N	AI
K	E	AK	P	R
U	F	L	V	AE
D	H	I	AF	Y
A	W	T	AG	Z
B	J	Q	AH	AJ
M			AD	AA
C				AB
				AL
				AM
				X
				AC

The same convergence data is generated as in previous test groups and, again, the results indicate that the model is successfully converging on better solutions. This is shown in Figure 22.

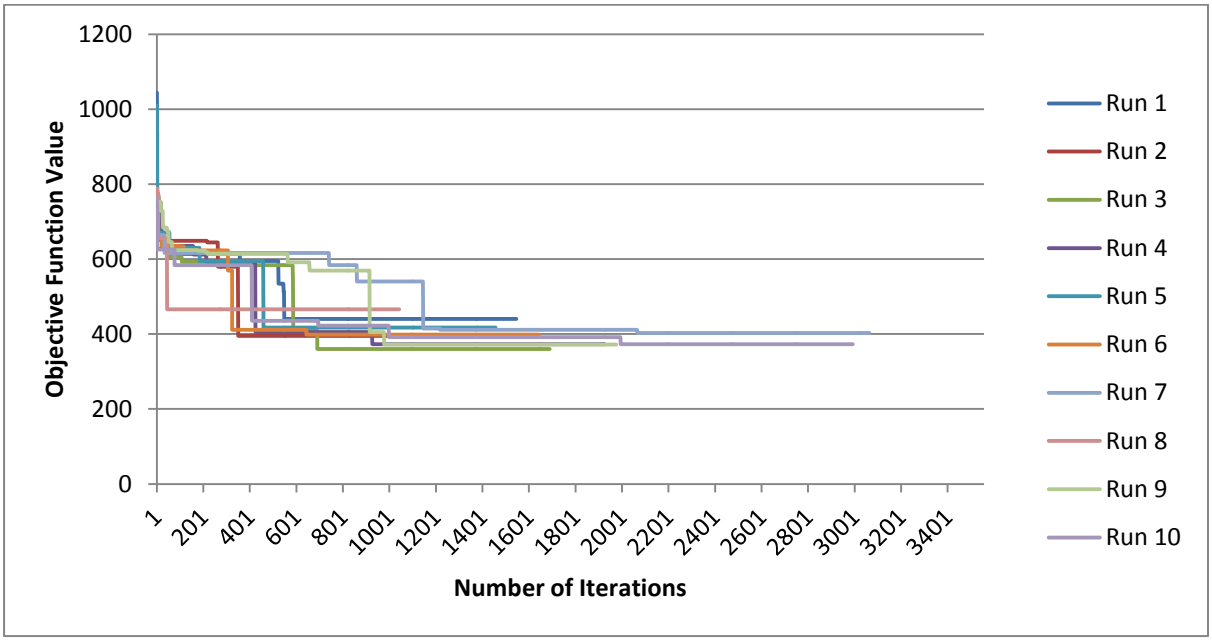


Figure 22: Test Group 3 Convergence Graph

The results which are obtained with regard to loading and work package data show a large increase in all three areas of loading with the average line loading increasing by 12.72%.

Average Line Loading: 78.69%

Average Max Loading: 91.52%

Average Min Loading: 72.72%

Number of Work Packages: 5

This increase is due to a single unit decrease in the number of work packages, which, in turn, leads the content to be distributed amongst the remaining work packages. The increased loadings are shown in Figure 23. All work packages have maximum loadings between 80% and 100%. Despite the unbalanced average utilisations, there is an improvement in efficiency as a result of the increased takt time.

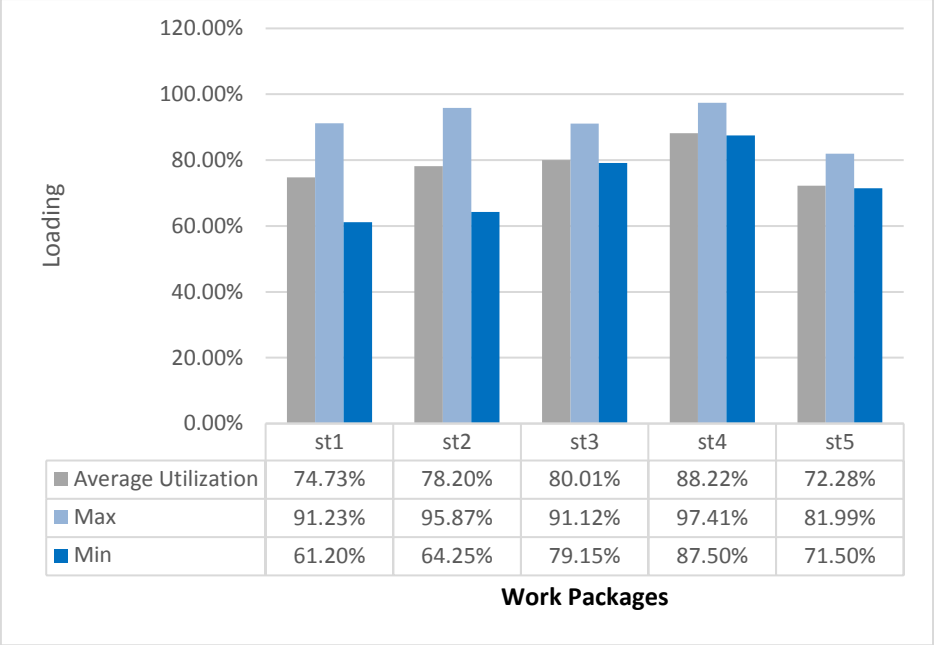


Figure 23: Test Run 3 Best Solution Min/Max Graph

The improvements in loading as a result of the removal of a work package translate into an improved level of stability in the line, since it addresses the issue of under loading in previous test groups. This is shown in Figure 24 where, in contrast to previous test groups, the stability shows an improvement, with station 4 indicating all units being loaded above 85%.

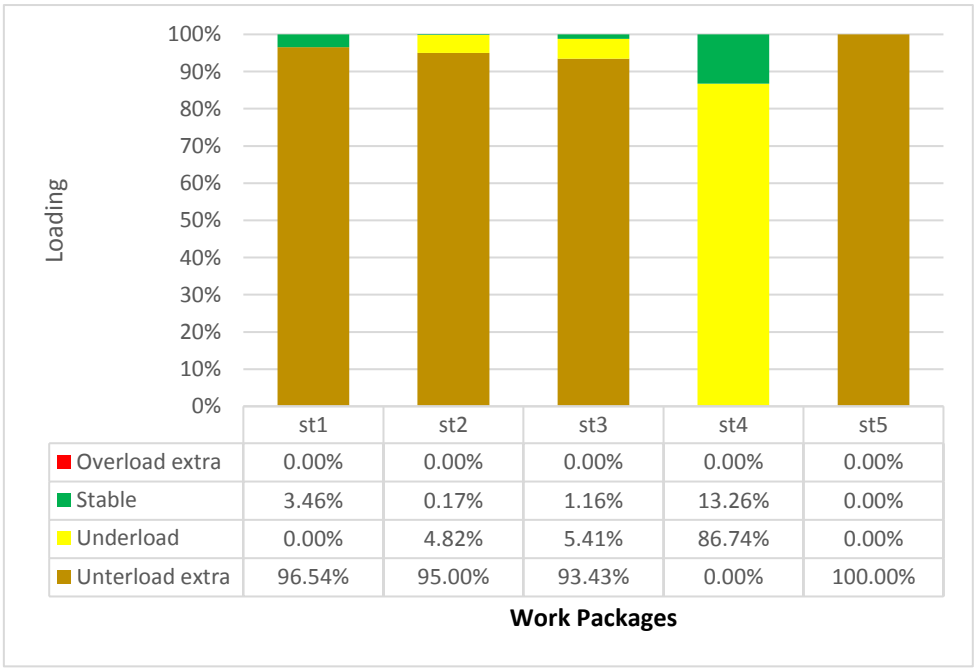


Figure 24: Test Group 3 Best Solution Stability Graph

The results for test group 3 indicate that there is an increase in efficiency as a result of increasing the takt time in the model without adding infeasible work packages. The only problem is that the results and solutions violate an exclusivity constraint.

10.4.4 Test Group 4

In order to address the penalty incurred in test group 3, the sequence data was changed for a second time in order to allow the algorithm to select a viable process without violating this constraint. This change can be seen in Appendix C, where it is represented as the second modification in constraints. Half of the process steps which form part of the exclusivity constraints were moved closer towards the start of the line. This separates the processes in the exclusivity constraints and allows the algorithm to place the process step in a different work package.

The fourth test group incorporated these exclusivity constraint changes. The fourth test group used the following inputs:

Takt Time: 252s

Pheromone deposit rate: 3%

Pheromone evaporation rate: 0.2%

The best solutions from all ten runs from test group 4 are shown in Table 11. The objective functions of this test group are much lower than those of previous groups and do not have penalties. This indicates that the exclusivity constraint changes made were successful in eliminating the penalties obtained in the third test group's solutions. It also indicates solutions which are significantly more efficient than any in previous test groups.

Table 11: Best Solutions from Test Group 4

Runs	Best Obj	Penalties
1	173.6893	0
2	124.961	0
3	143.5599	0
4	168.4857	0
5	167.4861	0
6	151.4981	0
7	150.9261	0
8	132.6764	0
9	147.0471	0
10	132.6764	0

As in previous test groups, the convergence graph shown in Figure 25 indicates that the model is still converging consistently towards good solutions. What is notable Figure 25 and in Figure 22 (in test group three) is the point in time where the solution's objective function suddenly drops by over 200. This corresponds to the point where the model first finds a solution which only requires five work packages. This means that the objective function will drop by an amount equal to the takt time, since the sum of the waiting time in the solution will have dropped by this amount.

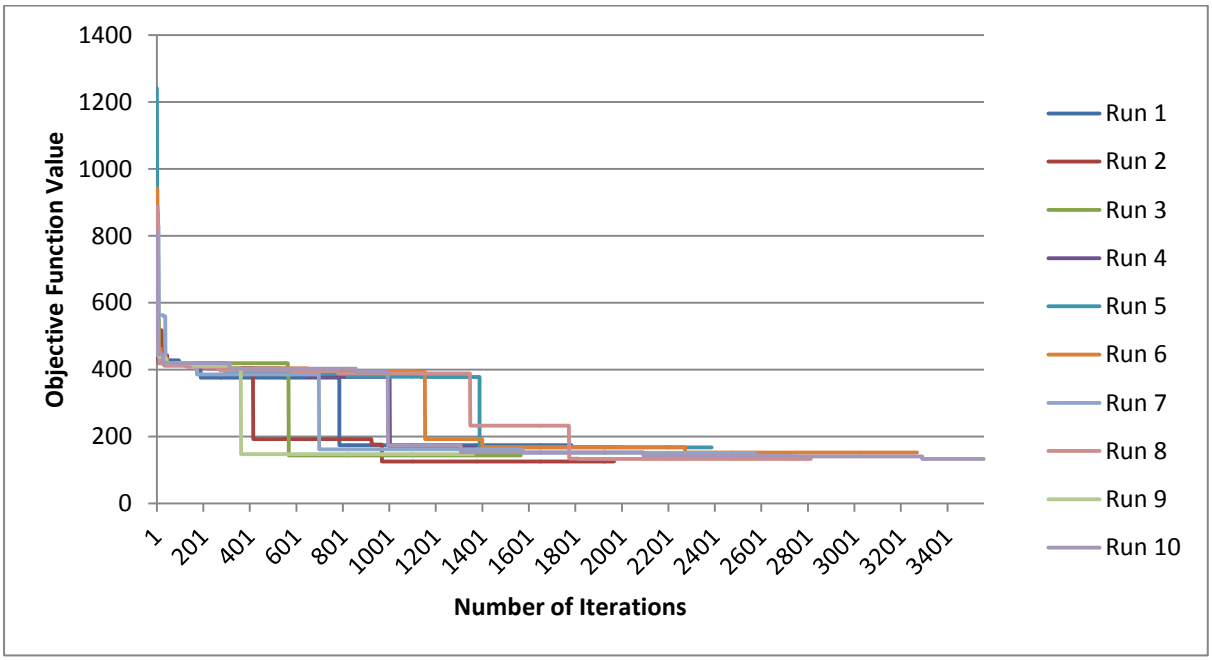


Figure 25: Test Group 4 Convergence Graph

The loading results of test group four (Figure 26) shows numbers which are almost identical to those of test group three. However, test group four does not violate any

exclusivity constraints. These are therefore feasible solutions with the same efficiency gains as test group 3. The average loading results are as follows:

Average Line Loading: 78.69%

Average Max Loading: 91.81%

Average Min Loading: 72.72%

Number of Work Packages: 5

The detailed loading results from each work package are shown in Figure 26, where the maximum loadings of all the work packages are located above 80% and the average utilisations of all the work packages are above 60%. The only area of concern is in station 2 where the maximum utilisation goes above 100% to 106.28%. This is due to the takt time being increased. To determine whether this will be a problem in the production process, further analysis of the stability data is needed.

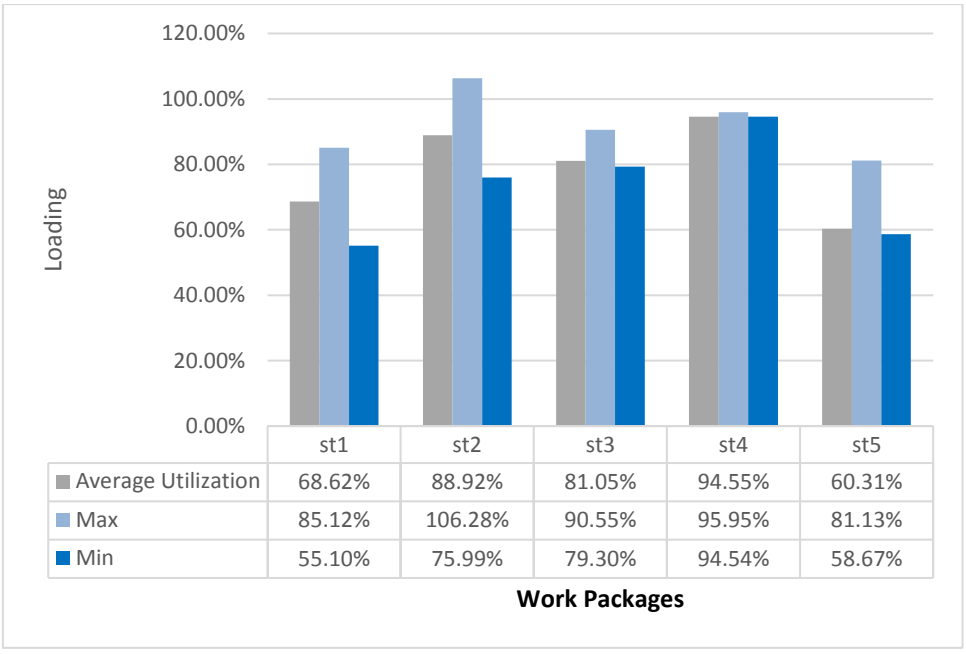


Figure 26: Test Group 4 Best Solution Min/Max Graph

The stability data results for test group 4 are shown in Figure 27. The improvements in these test results are clearly evident. There are three work packages which reach the stable range, with station 4 being entirely stable. This is the ideal state for a work package. There is only one work package which is under loaded, this being station 5. The overloading issue in station 2 which was apparent in Figure 26 can be seen in the stability graph. The percentage of production volume which is affected only amounts to 0.17% and is a relatively small proportion. At the plants present

production volumes, on average, there will be an overload situation every two days. This is acceptable, especially since the overloading is small at 106% loading.

Figure 27: Test Group 4 Best Solution Stability Graph

The best solution obtained from test group four is an effective solution, and shows an improvement from the current state. Additionally, the sum total of available time (waiting times) in all the packages amounts to 40.97%, which indicates that any further work package reductions would be impossible without experiencing major overloading. Therefore this solution is the best solution obtained.

10.4.5 Final Solution

The final solution is the best result from test group four. It shows good efficiency and stability metrics. This solution had the lowest objective function of all solutions obtained in all test runs at 125.0. This is due to the solution's high efficiency, balanced loading across work packages, and it's adherence to all constraints. The work package structure of the solution can be seen in Table 12. This can be compared to the network diagram in Appendix C. It follows the process sequence correctly. The detailed steps of how the frontend of the vehicle is built using these process steps are shown in Appendix J. These detailed steps are what will be proposed to the team at the assembly line for line improvement.

Table 12: Final Solution Process Steps

Station 1	Station 2	Station 3	Station 4	Station 5
D	F	P	T	Q
K	C	X	I	R
G	O	AD	U	AI
A	E	AE	V	Y
B	H	AK	AF	AB
M	J	W	AG	AA
	S	N	AH	AL
		L		Z
				AJ
				AM
				AC

The work package times for the above work packages, obtained from the model, are shown in Table 13. All the packages are below the adjusted takt time of 252s. The small differences in loading between these and Figure 26 are attributed to minor changes in process time over the course of the study due to re-analysis of current processes. These time studies occur continuously as process steps change in small ways throughout the lifecycle of the product.

Table 13: Final Solution Process Times

	Station 1	Station 2	Station 3	Station 4	Station 5
Maximum	245.04s	251.28s	232.14s	223.92s	241.98s
Average	190.38s	192.34s	190.92s	219.27s	161.95s

In Table 14, the work areas of the final solution are shown with A indicating a process which can be done from the front or rear of the assembly, V indicating a process that is done at the front of the assembly and H indicating a process which takes place at the rear of the assembly. This means work packages 1, 2, 3 and 5 are done from the front of the assembly piece, and work package 4 is done from the rear of the assembly piece.

Table 14: Final Solution Work Area List

Station 1	Station 2	Station 3	Station 4	Station 5
A	A	A	A	V
A	A	V	A	V
A	V	V	A	A
A	V	A	A	V
V	V	A	H	V
V	A	V	H	V
	A	A	H	V
		V		V
				A
				V
				V

10.5 Discussion

10.5.1 Results

The result obtained from this study (page 62 and Appendix J) is a process which can be followed to build a frontend module for a 3-series sedan (F30). The resultant solution was compared to the current state of this area of the assembly line, and shows large improvements in efficiency and stability (Section 10.4.4). There are many benefits to these improvements, such as lower costs, higher quality and higher output. The lower cost of the production line is directly related to the decrease in the number of work packages. This translates to a reduction in the manpower requirements of the plant. The average cost of an associate on the production line is R 23 818.28 per month. Therefore the saving achieved by the reduction from six work packages to five work packages amounts to R 285 819.36 per annum. This is significant considering the length of the production line. It means that this line will be running at peak efficiency at the current takt time of 232 seconds, since no additional headcount reduction is possible. The improvements regarding quality and output result from the higher level of stability. Associates no longer have to rush and can work at a more consistent pace, leading to higher attention to work and less stoppages as a result of overloading. This means that there is likely to be an increase in output and a reduction in defects. This, however, is impossible to measure until the solution is implemented on the line. The summary of all the proposed solution improvements, when compared to the current solution, can be seen in Table 15.

Table 15: Summary of Final Solution Improvements

Metric	Current	Proposed	Improvement
Average Loading	65.57%	78.69%	**13.12%**
Average Maximum Loading	76.38%	91.81%	**15.43%**
Average Minimum Loading	60.60%	72.72%	**12.12%**
Number of Work Packages	6	5	**-1**
Average Stability	0.10%	17.91%	**17.81%**
Labour Cost (per annum)	R 1 714 916.16	R 1 429 096.80	**- R 285 819.36**

The final solution only specified the process steps. There are two more steps which need to be followed in order to be able to implement a solution. These are placing the work packages and the relevant equipment on the line, and taking the movement and delivery of parts into account. This is due to the fact that the structure of the line cannot be changed without great cost. The sequence constraints have already taken structural constraints into account. However, the ideal position for an associate to work needs to be determined. The movement of associates around the

line and the parts supply to each station for larger parts also needs to be considered. The layout corresponding to the final solution is shown in Figure 28. The assembly line for the frontend pre-assembly is shown with the locations of all the assembly work packages. The supply of large parts is done through external sequencing processes through car sets and, in the case of the bumpers and headlights, sequenced supply. The only internal parts movement outside of the movement of the work piece carriers is the movement of sub-assemblies from work package 1 to work package 2. This would be required to achieve the proposed line balancing solution. The work packages are all fixed in their stations with the exception of the third work package, which requires the associate to move to the fifth work package area to assist with the picking of the front and rear bumpers, as required by the exclusivity constraints. The benefit of using this layout is the avoidance of moving fixed equipment, such as the sub-assembly tables, as this would require a large investment. The equipment which does need to move for this implementation is mobile and therefore has no costs tied to its relocation.

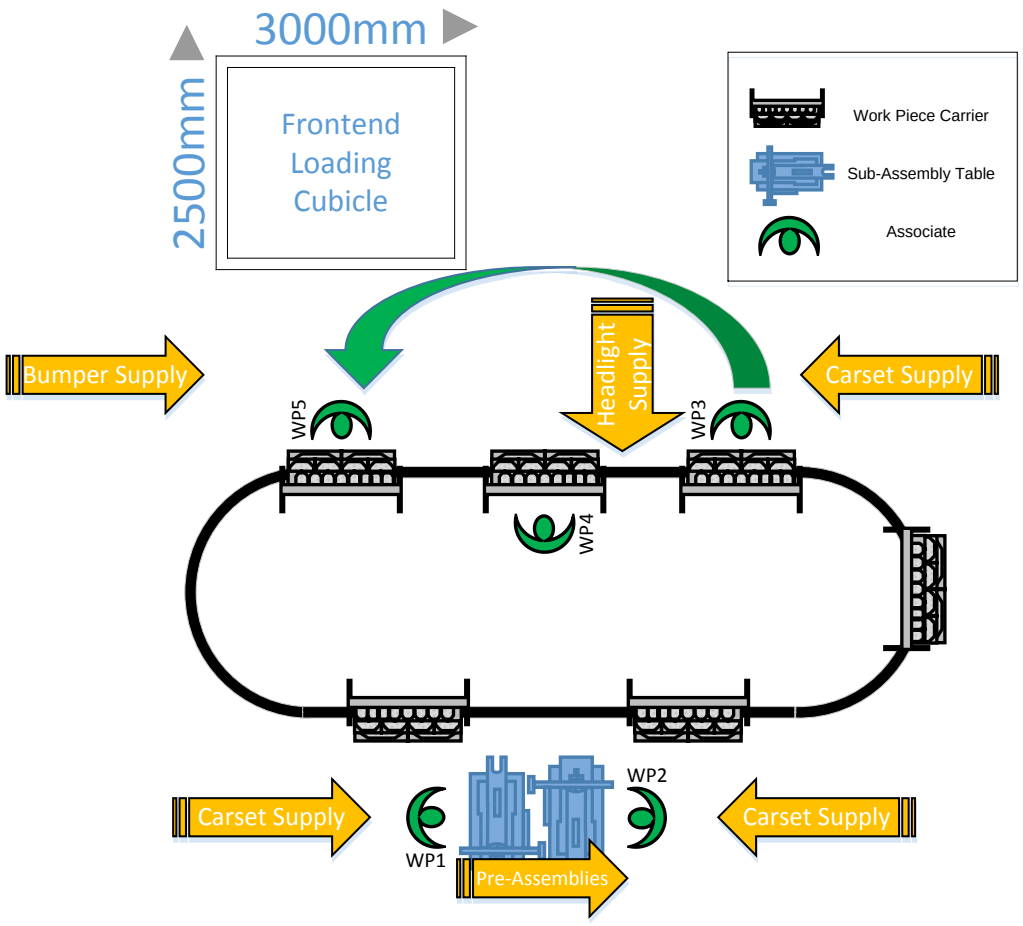


Figure 28: Assembly Line Final Solution Layout

Therefore the final solution is possible to implement with minimal investment and offers cost, quality and output improvements for the assembly line.

10.5.2 Model

The model used specifically designed for this sub-assembly line. If this model were to be used on another line with a different structure, modifications would be required.

The frontend assembly line problem follows the characteristics of a paced synchronous production line with sequential operations. This means that the problem follows a series of processes in a sequential order. For this reason the model used to solve the problem is designed to construct the solution in a sequential manner. If this model were used to solve another problem on another assembly line, where there are parallel processes, or simultaneous processes, then modifications to the algorithm would be required. The current algorithm has no built in functionality which takes these parallel processes into account, other than the exclusivity constraints. These exclusivity constraints require manual intervention in some cases, which would not be practical if there were many constraints of this type. The constraints fail to place the processes in any location related to one another, but simply avoid placing them in the same location. Parallel processes may be physically placed in incorrect locations on the line, resulting in infeasibility.

The second limitation of the model is the inability of the algorithm to consider the layout of the production line. The exclusive processes had to be manipulated manually through the modification of the sequence constraints in order to get the model to assign these processes to workable locations. If the constraints were left in place, the model would have consistently found infeasible solutions. For this reason, an improvement to the algorithm would be a placement routine where the algorithm, instead of generating a sequence, places processes in locations so that any processes which require the simultaneous work of two operators can be placed simultaneously. This would render the exclusivity constraints irrelevant, and would provide more implementable solutions in a production process which has many simultaneous operations.

The algorithm created is one which solves problems in a short amount of time. However there is a large amount of data capturing and manipulation required prior to its use. The preparation phase of the algorithm is summarised in Figure 29, where the process of data preparation and the points at which the data is captured from the raw data from the BMW South Africa systems can be seen. What is apparent in Figure 29 is that there is an extensive amount of once-off data preparation required. Many of these data preparation steps require an in depth knowledge of the production process. They are time consuming, and lead to an increase in the time to generate a solution. In addition, when the algorithm is run many times in

succession, such as in this study, the data input process becomes the most time consuming part process. Therefore two improvements in data preparation are:

1. Streamline the data preparation phase by integrating the required data into the core process step structure. This will allow for the bulk of the work to be done at the launch of a new vehicle, instead of every time the process needs to be optimised.
2. Modify the algorithm to run successive iterations of the algorithm while automatically changing the takt time and pheromone rates. This would mean that the entering of input data would only be required once at the start of the process, instead of the start of every algorithm iteration.

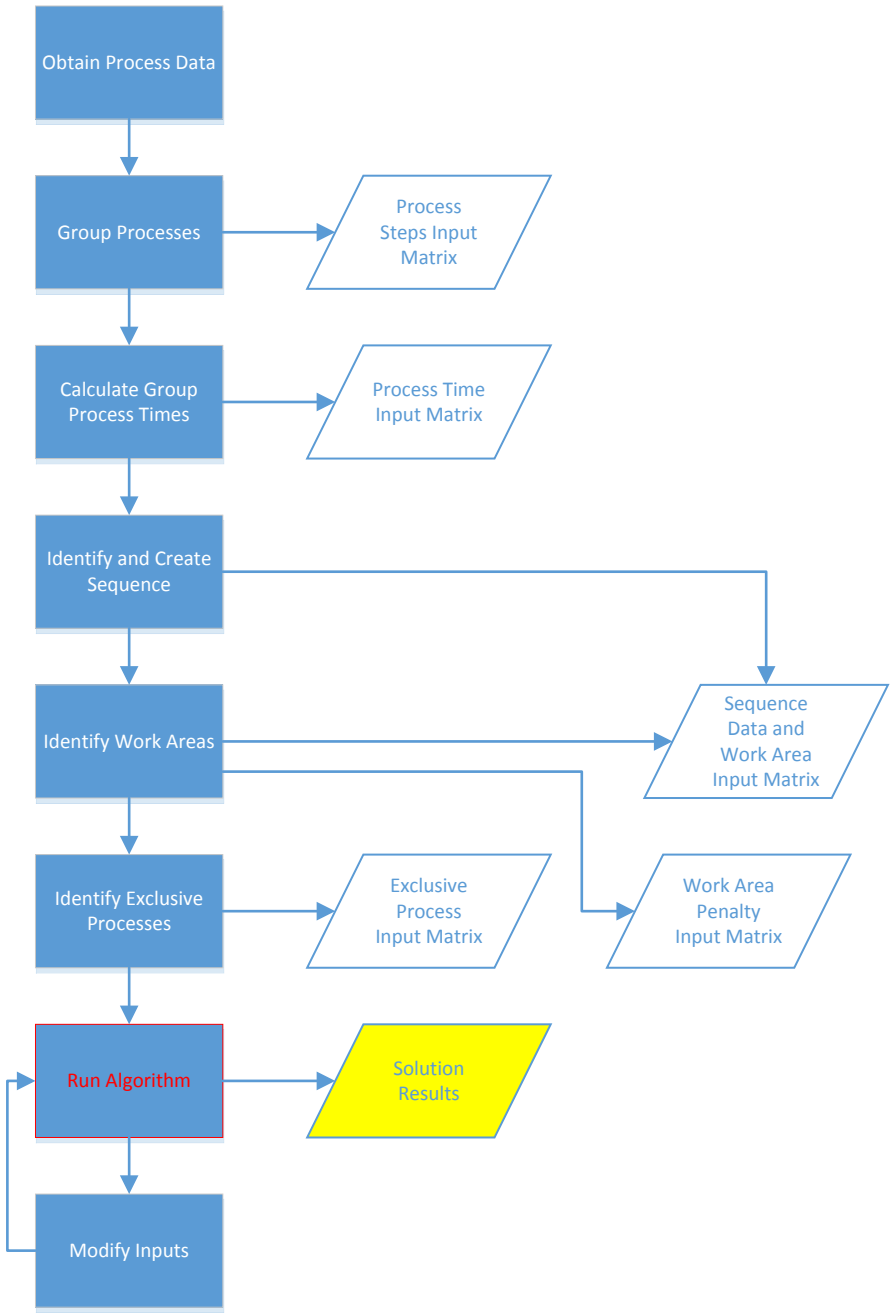


Figure 29: Data Preparation Process Flow

The last step in Figure 29 (Modify Inputs) refers to the iterative process in section 10.4 where the takt time, pheromone rates and work area penalties were modified. These processes were done through trial and error. The setting of work area penalties is essential and needs to be done to the analyst's best judgement, according to the severity of working between two work areas.

Exclusivity penalties which are violated by the solution increase the objective function by an amount which is equal to the takt time. The purpose of this is to make such a solution equal to another solution which requires an additional operator. This

is because two exclusive processes are allocated to a single work package, then such a solution would require an additional associate to assist with this process. The penalties applied to violating work areas are left to the discretion of the analyst. The logic behind setting these is to consider how much additional time the movement between work areas would take, and what the ergonomic and safety risk would be in doing so. In the frontend sub-assembly example the movement between the front and rear of the carrier requires crossing the railing carrying the work piece carrier and this is a safety risk. Therefore the movement between these areas is limited.

The final limitation of the algorithm regards the analysis process. A large component of the analysis of the suitability of any solution requires the use of a line balancing tool. This compares the line balancing solution with historical order data to obtain the loading and stability data seen in section 10.4. This process involves converting a solution into data suitable for the line balancing tool. This is time consuming and leads to duplication of work, which the algorithm is already doing in the form of placing process steps in a sequence. Therefore a possible improvement is to either incorporate line balancing software into the algorithm, or to provide an output solution result which can be easily input into the line balancing software.

11 Conclusions

The solution obtained from the model is implementable physically, can be expected to save R 285 819.36 per annum and improve the quality and output of the area. This will be done by achieving improvements in the loading and stability of the work packages. There will be a decrease in the amount of rework required and there is a lower likelihood of causing stoppages on the main production line. These lead to a loss of overall volume. This improvement was made with minimal investment being required, as the solution matches the current line structure.

Despite the fact that an improvement was made, there are a number of ways in which the algorithm can be improved for use in other areas of the assembly plant. These improvements are as follows:

- The algorithm can be modified to place work packages in specific locations, instead of just generating a sequential process. This would allow the algorithm to provide more implementable solutions, and would provide better solutions where there are processes which require operators to work together.
- Streamline the data preparation phase by both reducing the number of inputs required for the algorithm, and by integrating the data required into the existing TVG and work package structure. Then the data capturing process can take place as part of the standard process development and vehicle launch process.
- Modify the algorithm to run successive iterations with automated changes to the input parameters. This would reduce the manual data processing time.
- Integrate the analysis software for loading and stability into the algorithm itself, or provide an output suited to existing line balancing software.

Through these improvements, the algorithm could become a tool which could effectively be used to provide initial line balancing solutions for new model implementations, as well as optimisations to existing line balancing problems in any assembly operation. This could lead to improvements in efficiency, quality and output of the production process.

12 References

1. **Womack, Jim and Jones, Dan.** *Lean Thinking.* New York : Free Press, 2003. ISBN 0-7432-4927-5.

2. *Balancing parallel two-sided assembly lines.* **Ozcan, Ugur, Gokcen, Hadi and Toklu, Bilal.** 16, 15 August 2010, International Journal of Production Research, Vol. 48, pp. 4767–4784.

3. *A bi-objective metaheuristic approach to unpaced synchronous production line-balancing problems.* **Chiang, Wen-Chyuan, Urban, L. Timothy and Xu, Xiaojing.** 1, Tulsa : International Journal of Production Research, 2011, Vol. 50.

4. **Grzechna, Waldemar.** Final Results of Assembly Line. *Assembly Line - Theory and Practice.* s.l. : Intech, 2011, pp. 3-12.

5. **Uddin, Mohammad Kamal and Lastra, Jose Luis Martinez.** Assembly Line Balancing and Sequencing. *Assembly Line - Theory and Practice.* s.l. : InTech, 2011, pp. 13-36.

6. *Triumph of the Lean Production System.* **Krafcik, John.** 1, s.l. : Sloan Management Review, 1988, Vol. 30, pp. 41-52.

7. **Akinlawon, Akin O.** Thinking of Lean Manufacturing Systems. *SAE International.* [Online] [Cited: 29 December 2016.] http://www.sae.org/manufacturing/lean/column/leandec01.htm.

8. **Strategos International.** All About Takt Time. *Strategos Consulting.* [Online] [Cited: 29 December 2016.] http://www.strategosinc.com/takt_time.htm.

9. **Six Sigma Material.** Line Balancing. *Six Sigma Material.* [Online] [Cited: 29 December 2016.] http://www.six-sigma-material.com/Line-Balancing.html.

10. **Groover, Mikell P.** *Work Systems and Methods, Measurement, and Management of Work.* s.l. : Pearson Education International, 2007.

11. *Two-sided assembly line balancing: a genetic algorithm.* **Kim, Y.K., Kim, Y. and Kim, Y.J.** 1, 2000, Production Planning and Control, Vol. 11, pp. 44–53.

12. **Hillier, Frederick S. and Lieberman, Gerald J.** *Introduction to Operations Research Eighth Edition.* New York : McGraw-Hill, 2005. ISBN 0-07-252744-7.

13. **Rama Murthy, P.** *Operations Research.* New Dehli : New Age Internation (P) Ltd. Publishers, 2007. ISBN: 978-81-224-2944-2.

14. **Glover, Fred and Kochenberger, Gary A.** Preface. *Handbook of Metaheuristics.* s.l. : Kluwer Academic Publishers, 2003, pp. xi-xii.

15. **Dorigo, Marco and Stutzle, Thomas.** The Ant Colony Optimisation Metaheuristic: Algorithms, Applications, and Advances. [ed.] Fred Glover and Gary A. Kochenberger. *Handbook of Metaheuristics.* s.l. : Kluwer Academic Publishers, 2003, pp. 251-285.

16. **Reeves, Colin.** Genetic Algorithms. [ed.] Fred Glover and Gary A. Kochenberger. *Hanbook of Metaheuristics.* 2003, pp. 55-82.

17. **Gendreau, Michel.** An Introduction to Tabu Search. [ed.] Fred Glover and Gary A. Kochenberger. *Handbook of Metaheuristics.* s.l. : Kluwer Academic Publishers, 2003, pp. 37-54.

18. *Tabu Search—Part I.* **Glover, F.** 1, 1989, ORSA Journal on Computing, pp. 190–206.

19. **Henderson, Darrall, Jacobson, Sheldon H. and Johnson, Alan W.** The Theory and Practice of Simulated Annealing. [ed.] Fred Glover and Gary A. Kochenberger. *Handbook of Metaheuristics.* s.l. : Kluwer Academic Publishers, 2003, pp. 287-319.

20. **Lorena, Luiz A. N., Narciso, Marcelo G. and Beasley, J. E.** A Constructive Genetic Algorithm For The Generalised Assignment Problem. *Laboratório Associado de Computação e Matemática Aplicada.* [Online] 2000. http://www.lac.inpe.br/~lorena/gap/CGA-PGA-2000.pdf.

13 Appendices

13.1 Appendix A: Sample Process Data for Build in Frontend Line

Table 16: Sample of TVG Data for Assembling a Frontend

Type	Tact Id	Worker area	Seq	TVG Id	TVG description	Number	Total Time
ZH	9001	P	10	Z 5111 003 035 A 03	SCAN OPTION LIST (DEFOBOX)	109	3.42s
ZW	9001	P	20	Z 5111 003 005 A 01	WALK TO SUPPLY TROLLEY AND BACK (DEFO BOX AND LOWER BRACKETS)	109	3.60s
M	9001	P	30	Z 5111 003 A0R A 09	PREASSY 1 DEFO BOX 1 + 2 LP	109	29.58s
M	9001	P	40	Z 5111 003 A2R A 01	PREASSY 1 DEFO BOX	91	18.42s
ZW	9001	P	50	Z 5111 003 025 A 02	WALK TO SUPPLY TROLLEY AND BACK (AIR GUIDE)	109	3.60s
M	9001	P	60	Z 5174 002 25R A 01	PREASSY 1 AIR GUIDE LOWER TO DEFO BOX	109	4.68s
ZW	9001	P	70	Z 5111 003 015 A 01	WALK TO SUPPLY TROLLEY AND BACK (LOWER BUMPER BEAM)	109	1.80s
M	9001	P	80	Z 5111 003 21R A 10	PREASSY 1 CROSSMEMBER 2ND LOADPATH TO DEFO BOX PUT FASTEN	91	8.28s
M	9001	P	90	Z 5111 003 25R A 05	PREASSY 1 CROSSMEMBER 2ND LOADPATH TO DEFO BOX	18	8.64s
ZW	9001	P	100	Z 5111 003 010 A 01	WALK TO SUPPLY TROLLEY AND BACK (CROSSMEMBER)	109	1.80s
M	9001	P	110	Z 5111 003 30R A 06	PREASSY 1 CROSSMEMBER 1ST LOADPATH TO DEFO BOX	109	28.32s
M	9001	P	120	Z 5111 003 22R A 02	PREASSY 1 CROSSMEMBER 2ND LOADPATH TO DEFO BOX SCREWS TIGHTEN	91	6.12s
M	9001	P	130	Z 5111 003 24R A 01	PREASSY 1 CROSSMEMBER 2ND LOADPATH TO DEFO BOX	18	6.12s
M	9001	P	140	Z 5111 003 50R A 03	PREASSY 2 ASSY SHOCK ABSORBER FITMENT	109	6.12s
ZH	9001	P	150	Z 5164 001 005 A 01	RELEASE FIXTURE AND WALK TO CONVEYOR	109	4.50s
M	9001	P	160	Z 5111 003 40R A 03	PREASSY 1 ASSY CRASH SYSTEM FROM PREASSY TABLE IN HYDR/VEH	109	6.30s
ZH	9001	P	170	Z 5111 003 250 A 01	PREASSY2 ASSY FIT SHOCK ABSORBER #ADD/HANDL	109	3.60s
ZW	9001	P	180	Z 5111 003 030 A 01	WALK TO REPLACE AN EMPTY TROLLEY TO A FULL ONE	109	3.66s
M	9001	P	190	Z 5111 003 80R A 05	Scanning Process ZB crashsystem subassy	109	4.86s

13.2 Appendix B: Build sequence for build in frontend line

Table 17: Process Grouping for Sequencing Processing

Process Number	Description	Sub-Processes
A	Sub-Assembly 1 Preparation	Scanning
B	Sub-Assembly 1	Lower Defobox into Jig
		Upper Defobox into Jig
		Aligning Defoboxes
		Tightening Defoboxes
		Fit Lower Beam to Defobox
		Fit Upper Beam to Defobox
		Release Defoboxes and Carry to Carrier
		Lock Carrier
C	Lower Air Guide	Clip Lower Air Guide to Defoboxes
D	ACC Sub-assembly	Sub-assemble ACC to Bracket
E	ACC Fitment	Fit ACC to Frontend
F	Fit Foam Absorber	Fit Foam Absorber to Upper Beam
G	Fit Dust Cap to Lock Support	Fit Dust Cap to Lock Support
H	Sub-Assembly 2	Fit Lock Support Assembly into Jig
		Fit Crossmember into Jig
		Fit Locks into Jig
		Pre-tighten Locks
		Assemble Upper Air Guide
		Pre-tighten Upper Air Guide to Cross Connection
		Lock Jig for Tightening
		Tighten Locks
		Unlock Jig
		Carry Assembly to Carrier
		Place Assembly on Carrier
		Lock Fixture on Carrier
I	Pre-fit Headlight Brackets	Fit Headlight Brackets
J	Fit and Tighten US Crash Sensors	Fit and Tighten US Crash Sensors
K	Sub-assemble V-bar to Air Guide	Sub-assemble V-bar to Air Guide
L	Tighten Lock Support to Defobox	Tighten Lock Support to Defobox
M	Fit Kerb Dampers	Fit Kerb Dampers
N	Fit Horns	Fit Horns
O	Fit Brake Air Guides	Fit Brake Air Guides
P	Fit Upper Moek Bracket	Fit Upper Moek Bracket
Q	Fit Lower Moek Bracket	Fit Lower Moek Bracket
R	Fit Moek Air Guides	Fit Moek Air Guides
S	Tighten V-bar Sides	Tighten V-bar Sides
T	Fit Snorkel	Fit Air Guide RHS

		Fit Snorkel RHS
		Fit Grill/Cap RHS
		Fit Air Guide LHS
		Fit Snorkel LHS
		Fit Grill/Cap LHS
		Scanning
U	Place Tape on Headlights	Place Tape on Headlights
V	Position Headlights	Position Headlights
W	Fit Bumper Support Bracket	Fit Bumper Support Bracket
		Insert Clip to Crossmember
X	Fit Front Bumper to Frontend (Ass. 1)	Carry Front Bumper to Carrier (Ass. 1)
		Bumper Inspection (Ass. 1)
		Assemble Front bumper (Ass.1)
Y	Fit Front Bumper to Frontend (Ass. 2)	Carry Front Bumper to Carrier (Ass. 2)
		Bumper Inspection (Ass. 2)
		Assemble Front bumper (Ass.2)
Z	Connect Harness to ACC Sensor	Connect Harness to ACC Sensor
AA	Pre-tighten Front Bumper to Front End	Pre-tighten Front Bumper to Front End
AB	Fit Emblem	Fit Emblem
AC	Collect and Load Rear Bumper (Ass. 1)	Collect and Load Rear Bumper (Ass. 1)
		Inspect Rear Bumper (Ass. 1)
AD	Collect and Load Rear Bumper (Ass. 2)	Collect and Load Rear Bumper (Ass. 2)
		Inspect Rear Bumper (Ass. 2)
AE	Fit C-Clips to Upper Air Guide	Fit C-Clips to Upper Air Guide
AF	Pre-Tighten Lights	Pre-Tighten Lights
AG	Scan Headlights	Scan Headlights
AH	Front End Alignment with Fixture	Front End Alignment with Fixture
		Tighten Upper Air Guide to Cross Connection
		Tighten Headlights
		Release Fixture
AI	Clip Upper Air Guide Flaps to Bumper Beam	Clip Upper Air Guide Flaps to Bumper Beam
AJ	Clip Bumper Z-Support Clips to Frontend	Clip Bumper Z-Support Clips to Frontend LH, RH and Centre
AK	Fit and Route Lock Bowden Cable	Fit and Route Lock Bowden Cable
AL	Apply Tape to Bumper and Headlights	Apply Tape to Bumper and Headlights
AM	Remove Frontend from Carrier onto LAM	Remove Frontend from Carrier onto LAM

13.3 Appendix C: Network Diagram for Build Sequence and Work Area

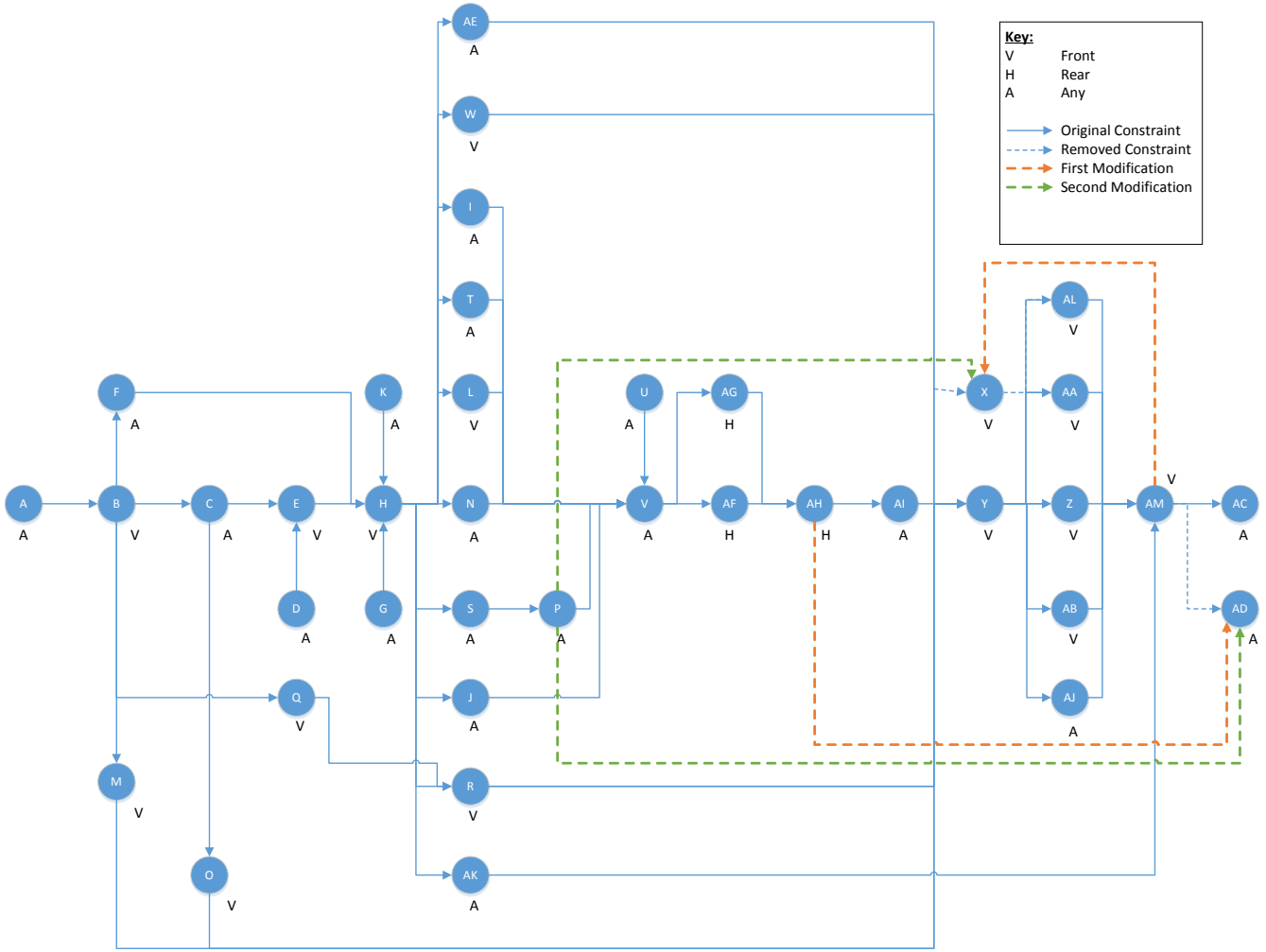


Figure 30: Network Diagram of Frontend Build Sequence

13.4 Appendix D: Example of Input Data for ACO Algorithm

13.4.1 Sequence Data and Work Area Input

Key:	
1	Any
2	Front
3	Rear

Table 18: Sequence Data and Work Area Input for ACO Algorithm

<- From ->	<- To ->																									
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	-1	0	1	0	0	1	0	0	0	0	0	0	2	0	0	0	2	0	0	0	0	0	0	0	0	0
C	0	-2	0	0	2	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	-1	-1	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F	0	-2	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	-2	-1	-1	0	1	1	-1	2	0	1	0	0	0	2	1	1	0	0	2	0	0	0
I	0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
J	0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
K	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
M	0	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
N	0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0			0
O	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0

<- From ->	<- To ->												
	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM
A	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	1	0	0	0	0	0	1	0	0
I	0	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0
O	0	0	0	0	0	0	0	0	0	0	0	0	0

	P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	1	0	2	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
	Q	0	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	R	0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
	S	0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	T	0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	V	0	0	0	0	0	0	0	0	-1	-1	0	-2	0	-1	0	-1	0	0	0	-1	-1	0	0	0	0	0	0	0	0	0	0	3	3	0	0	0	0	0	0
	W	0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Y	0	0	0	0	0	0	0	0	0	0	0	0	-2	0	-2	0	0	-2	0	0	0	0	-2	0	0	0	2	2	2	0	0	-1	0	0	0	-1	1	0	2	0
	Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
	AA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
	AB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
	AC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-2
	AD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	AE	0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	AF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0
	AG	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0
	AH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	-3	0	1	0	0	0	0
	AI	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	-3		0	0	0	0
	AJ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
	AK	0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
	AL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
	AM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-2	-2	-2	2	0	0	0	0	0	0	-1	-1	-2	0

13.4.2 Process Steps Input

Table 19: Process Step Input for ACO Algorithm

Step	Sequence
A	1
B	2
C	3
D	4
E	5
F	6
G	7
H	8
I	9
J	10
K	11
L	12
M	13
N	14
O	15
P	16
Q	17
R	18
S	19
T	20
U	21
V	22
W	23
X	24
Y	25
Z	26
AA	27
AB	28
AC	29
AD	30
AE	31
AF	32
AG	33
AH	34
AI	35
AJ	36
AK	37
AL	38
AM	39

13.4.3 Process Time Input

Table 20: Process Time Input for ACO Algorithm

Grouping	Total Time (s)	Average Time (s)
A	13.74	13.74
B	128.16	110.4176
C	3.6	3.6
D	42	5.084037
E	37.8	2.774312
F	9.72	9.72
G	4.68	4.68
H	124.56	124.56
I	16.56	16.56
J	13.68	10.04037
K	30.42	30.42
L	41.64	41.64
M	18.54	18.54
N	31.68	31.68
O	45.72	25.4444
P	50.1	10.77523
Q	52.38	7.059633
R	38.52	11.82055
S	16.2	16.2
T	61.2	57.96
U	7.5	7.5
V	20.16	20.16
W	6.3	5.485872
X	21.96	21.96
Y	21.96	21.96
Z	8.64	0.634128
AA	29.1	29.1
AB	8.94	8.94
AC	11.34	11.34
AD	22.14	22.14
AE	7.56	7.56
AF	46.44	46.44
AG	17.64	17.64
AH	61.92	60.50642
AI	4.5	4.5
AJ	4.26	4.26
AK	50.76	49.68
AL	7.5	7.5
AM	54.84	54.84

13.4.4 Work Area Input Data

Table 21: Work Area Input Matrix for ACO Algorithm

	A	V	H	M	U	L	R
A	0	0	0	100	100	100	100
V	0	0	100	100	100	100	100
H	0	100	0	100	100	100	100
M	100	100	100	100	100	100	100
U	100	100	100	100	100	100	100
L	100	100	100	100	100	100	100
R	100	100	100	100	100	100	100

13.4.5 Exclusive Process Input Data

Table 22: Exclusive Process Input Data

Process 1	Process 2
24	25
29	30

13.5 Appendix E: Work Area Data by Process

This data is input as part of the sequence data matrix.

Table 23: Work Areas for Every Process

Step	Work Area
A	A
B	V
C	A
D	A
E	V
F	A
G	A
H	V
I	A
J	A
K	A
L	V
M	V
N	A
O	V
P	A
Q	V
R	V
S	A
T	A
U	A
V	A
W	V
X	V
Y	V
Z	V
AA	V
AB	V
AC	A
AD	A
AE	A
AF	H
AG	H
AH	H
AI	A
AJ	A
AK	A
AL	V
AM	V

Key:	
A	Any
V	Front
H	Rear

13.6 Appendix F: Programming Logic Diagram

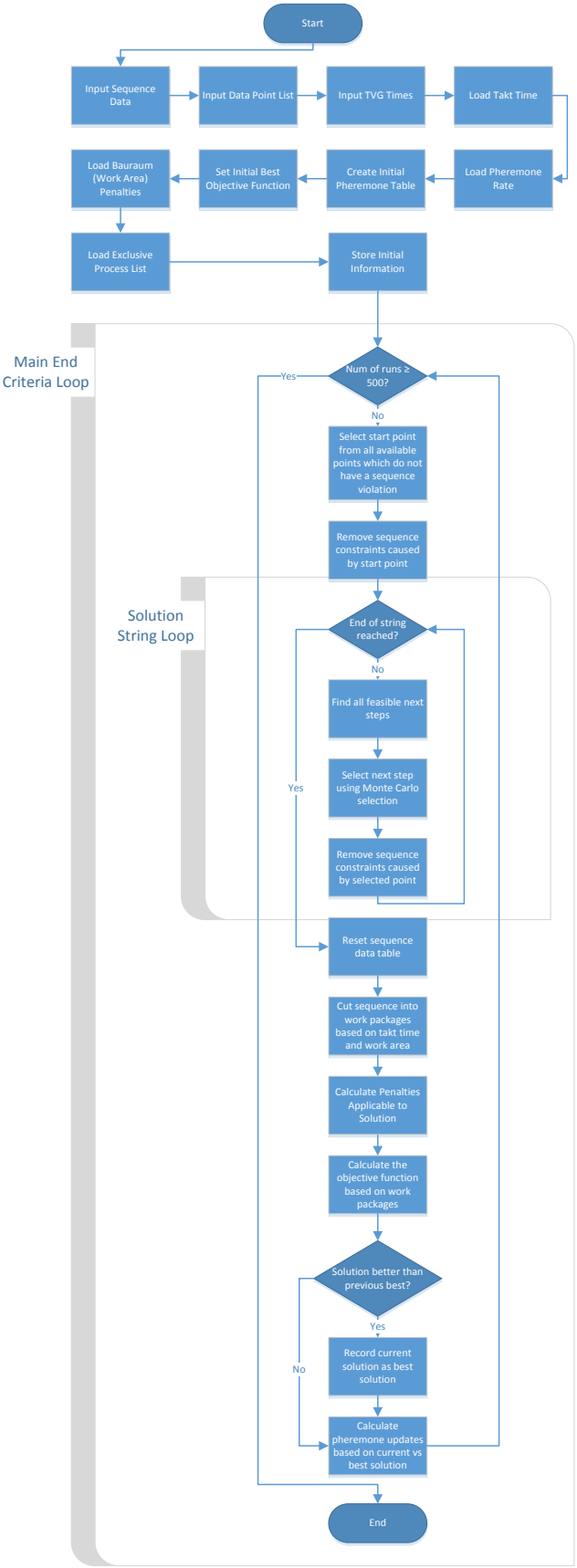


Figure 31: Algorithm Process Flow Diagram

13.7 Appendix G: Constrained ACO Algorithm in MATLAB Language

```
%This is version 6 of an Ant Colony Optimisation Algorithm for the purpose
%of balancing a takted assembly line.

clear

%Load the sequence data in matrix form
h = msgbox('Please select your sequence data file','Load Sequence
Data','help');
uiwait(h);
SequenceData = uiimport('-file');

%Load the list of data points in matrix form
h = msgbox('Please select your data point file','Load Step List','help');
uiwait(h);
StepList = uiimport('-file');

%Load the list of TVG times in matrix form
h = msgbox('Please select your step time file','Load Time List','help');
uiwait(h);
TVGTime = uiimport('-file');

%Load the takt time
Takt = inputdlg({'Enter Takt Time'}, 'Input',1,{'232'});
Takt = str2num(Takt{1,1});

%Load the Pheremone Rates
Tau = inputdlg({'Enter Pheremone Deposition Rate (~0.03)','Enter Pheremone
Evaporation Rate (~0.002)'}, 'Input',1,{'0.03','0.002'});
TauEv = Tau{2,1};
TauEv = str2num(TauEv);
Tau = str2num(Tau{1,1});

%Create initial pheremone table
PheremoneTable = ones(39);

%Create Initial Objective Function
BestObjectiveFunction = 9999999;

%Load the Bauraum Penalties Matrix
h = msgbox('Please select your Bauraum penalty file','Load Bauraum
List','help');
uiwait(h);
BauraumP = uiimport('-file');

%Load the Exclusive Process Penalty List
h = msgbox('Please select your exclusive process penalty file','Load
Exclusive Penalty List','help');
uiwait(h);
ExclusiveP = uiimport('-file');

Storage.SequenceData = SequenceData;
Storage.StepList = StepList;
Storage.TVGTime = TVGTime;
Storage.PheremoneTable = PheremoneTable;
```

```
SolutionReset = 1;
k = 1;

while SolutionReset < 1000
    clear BauraumList
    Penalties = 0;
%Look for start position - this searches for a start location which will
%not violate sequence initially and therefore it is a non-random start
%search methodology
    X = max(StepList.data);
    clear StartTest

for a = 1:X
        StartTest(a,1) = min(SequenceData.data(a,:));
end

    clear StartPositions
    StartPositions = find(StartTest >= 0);
    temp = ceil(rand(1,1)*length(StartPositions));
    Sol(1,1) = StartPositions(temp,1);

%Remove negative values in the sequence data matrix in order to release
%the algorithm to select any points which will no longer violate a sequence
constraint

for a = 1:X
if SequenceData.data(a,Sol(1,1)) < 0
            SequenceData.data(a,Sol(1,1)) = 0;
end
end

%Begin the loop to find the rest of the solution string
for j = 2:X

        clear WorkPackageTime
        clear WorkPackage
%Find ALL feasible next steps excluding steps which have been
%previously chosen
        t = 1;

for a = 1:X
if min(SequenceData.data(a,:)) < 0
else
if any(Sol == a) == 1
else
                    FeasibleSteps(1,t) = a;
                    t = t+1;
end
end
end

%MonteCarlo selection process for next step
        clear MonteCarlo;
        test = 1;
        t = length(FeasibleSteps);

for a = 1:t
if min(SequenceData.data(FeasibleSteps(1,a),:)) < 0
                MonteCarlo(1,a) = 0;
```

```
else
if test == 1
                    MonteCarlo(1,a)        =        PheremoneTable(Sol(1,(j-
1)),FeasibleSteps(1,a));
                    test = test + 1;
else
                    MonteCarlo(1,a)       =       MonteCarlo(1,a-1)       +
PheremoneTable(Sol(1,j-1),FeasibleSteps(1,a));
end
end
end

        MCRand = rand(1,1)*max(MonteCarlo);
            a = 1;
while MCRand > MonteCarlo(1,a)
                a = a+1;
end
        Sol(1,j) = FeasibleSteps(1,a);

%Remove negative values in the sequence data matrix in order to release
%the algorithm to select any points which will no longer violate a sequence
constraint
for a = 1:X
if SequenceData.data(a,Sol(1,j)) < 0
                SequenceData.data(a,Sol(1,j)) = 0;
end
end
        clear FeasibleSteps

end
    SequenceData = Storage.SequenceData;

%Cutting chosen sequence into work packages based on chosen takt time
    r = 1;
    c = 1;
    PrevBauraum = 1;
    PackageBauraum = 1;
    WorkPackageTime(1:2,1) = zeros;
for a = 1:X
        CurrentBauraum = max(abs(SequenceData.data(:,Sol(1,a))));

if PackageBauraum ~= 1
if CurrentBauraum ~= 1
if CurrentBauraum ~= PackageBauraum
if    rand(1,1)    <    (BauraumP.data(PackageBauraum,CurrentBauraum))/100
%used to be / by Takt
                        c = c+1;
                        PrevBauraum = 1;
                        r = 1;
                        WorkPackageTime(1:2,c) = zeros;
                        PackageBauraum = CurrentBauraum;
end
end
end
end

if (WorkPackageTime(1,c) + TVGTime.data(Sol(1,a),1)) > Takt
            c = c+1;
            PrevBauraum = 1;
            r = 1;
```

```
            WorkPackage(r,c) = Sol(1,a);
            WorkPackageTime(1:2,c) = zeros;
            WorkPackageTime(1,c)       =       WorkPackageTime(1,c)       +
TVGTime.data(Sol(1,a),1);
            WorkPackageTime(2,c)       =       WorkPackageTime(2,c)       +
TVGTime.data(Sol(1,a),2);
            BauraumList(r,c) = CurrentBauraum;
            PrevBauraum = CurrentBauraum;
            PackageBauraum = PrevBauraum;
            r = r+1;
else
            WorkPackage(r,c) = Sol(1,a);
            WorkPackageTime(1,c)       =       WorkPackageTime(1,c)       +
TVGTime.data(Sol(1,a),1);  %NOTE: The column of TVGTime Should be 1 for max
and 2 for average
            WorkPackageTime(2,c)       =       WorkPackageTime(2,c)       +
TVGTime.data(Sol(1,a),2);
if CurrentBauraum ~= PackageBauraum
if CurrentBauraum ~= 1
if PackageBauraum == 1
                        PackageBauraum = CurrentBauraum;
else
                        Penalties          =          Penalties          +
BauraumP.data(PackageBauraum,CurrentBauraum);
                        PackageBauraum = CurrentBauraum;
end
end
end
            BauraumList(r,c) = CurrentBauraum;
            PrevBauraum = CurrentBauraum;
            r = r+1;
end

end

%Calculating the objective Function based on the work packages
    NumWP = length(WorkPackageTime);

for a = 1:NumWP
        DeltaTAve(1,a)             =             abs(WorkPackageTime(2,a)-
mean(WorkPackageTime(2,:)));
end

for a = 1:NumWP
        DeltaTMax(1,a) = Takt - WorkPackageTime(1,a);
end

%Calculate Exclusivity Penalties
    a = size(ExclusiveP.data);
    ExclusiveLength = a(1,1);

for a = 1:ExclusiveLength
for b = 1:NumWP
            ExcTestA = find(WorkPackage(:,b)==ExclusiveP.data(a,1));
            ExcTestB = find(WorkPackage(:,b)==ExclusiveP.data(a,2));
            ExcTest = sum(ExcTestA)*sum(ExcTestB);
if ExcTest > 0
                Penalties = Penalties + Takt;
end
end
```

```
end

    ObjectiveFunction = (sum(DeltaTMax)+sum(DeltaTAve))+Penalties;
    clear DeltaTAve
    clear DeltaTMax

%Checking if a new best has been found and recording solutions for
%graphical outputs
if ObjectiveFunction < BestObjectiveFunction
        BestObjectiveFunction = ObjectiveFunction;
        BestSolution = WorkPackage;
        BestWorkPackagetime = WorkPackageTime;
        BestPenalties = Penalties;
        BestBauraumList = BauraumList;
        SolutionReset = 1;
end

        SolList(k,:) = Sol(1,:);
        PenaltiesList(k,:) = Penalties;
        ObjectiveFunctionList(k,:) = ObjectiveFunction;
        BestObjectiveFunctionList(k,:) = BestObjectiveFunction;

%Calculating pheremone updates based on current solution vs best
%solution
    PheremoneTable = PheremoneTable*(1-TauEv);

for a = 2:X
        PheremoneTable(Sol(1,a-1),Sol(1,a))    =    PheremoneTable(Sol(1,a-
1),Sol(1,a)) + ((BestObjectiveFunction/ObjectiveFunction)*Tau);
end

    clear BauraumTest
    clear Sol
    SolutionReset = SolutionReset + 1;
    k = k+1;
end
```

13.8 Appendix H: Result Export Code

```
filename = 'testdata.xlsx';
xlswrite(filename,BestBauraumList,1)
xlswrite(filename,BestObjectiveFunction,2)
xlswrite(filename,BestPenalties,3)
xlswrite(filename,BestSolution,4)
xlswrite(filename,BestWorkPackagetime,5)
xlswrite(filename,PheremoneTable,6)
xlswrite(filename,SolList,7)
xlswrite(filename,Takt,8)
xlswrite(filename,ObjectiveFunctionList,9)
xlswrite(filename,BestObjectiveFunctionList,10)
xlswrite(filename,Tau,11)
xlswrite(filename,TauEv,12)
clear all
```

13.9 Appendix I: Algorithm Nomenclature

BauraumList – The matrix containing all the work areas corresponding to the current solution in *WorkPackage*.

BauraumP – A matrix input by the user which contains the penalties for crossing between work areas in a single work package.

BestBauraumList – The list of work areas corresponding to *BestSolution*.

BestObjectiveFunction – The best objective function found so far in the model.

BestObjectiveFunctionList – The list of the all current best solutions over time. Rows are solutions.

BestPenalties – The sum of penalties incurred by the current best solution.

BestSolution – The current best solution found at any point of time.

c – The current column (work package) being used in the work package generation process.

CurrentBauraum – The current work area in the package generation process.

DeltaTAve – The sum of all the differences between the individual average work package time and the average of all the work package times. Determined from *WorkPackageTime*.

DeltaTMax – The sum off all the differences between the maximum work package time and the takt time. Determined from *WorkPackageTime* and *Takt*.

ExclusiveLength – The length of the *ExclusiveP* matrix.

ExclusiveP – The matrix containing the list of which steps cannot be placed together in a single work package.

ExcTest – The comparison of *ExcTestA* and *ExcTestB* to determine whether the two exlusive processes are in the same package.

ExcTestA – The variable used to find the first process out of a pair of processes which cannot share a package.

ExcTestB – The variable used to find the second process out of a pair of processes which cannot share a package.

FeasibleSteps – The matrix of all feasible next steps determined from the *SequenceData* matrix, used for Monte Carlo generation and selection.

j – The current point in the solution string which is being generated.

k – The count of the total number of solutions generated.

MCRand – Random number scaled to the size of *MonteCarlo* matrix to select next step from this matrix.

MonteCarlo – The matrix representing the Monte Carlo selection table for step selection. Generated using the *FeasibleSteps* and *PheremoneTable* matrices.

NumWP – The number of work packages generated from current solution. Determined from *WorkPackageTime* matrix.

ObjectiveFunction – The measure of the effectiveness of the current solution. Calculated from *DeltaTMax* and *DeltaTAve* and *Penalties*.

ObjectiveFunctionList – The list of all objective function values found in matrix form. Rows are solutions.

Penalties – The sum of all work area and exclusive process penalties which have been accumulated by the current solution

PheremoneTable – The current pheremone table stored in matrix format which is used for the Monte Carlo selection process.

PackageBauraum – The work area of the current work package, used to represent the work area of the associate at the current stage in the package generation process.

PrevBauraum – The work area of the previous step in the work package generation process.

R – The current row (work step) being used in the work package generation process.

SequenceData – The sequence data matrix input by the user in matrix format, contains both sequence information as well as work area information.

Sol – The current/most recent solution string.

SolList – The list of all solutions found in matrix form. Rows are solutions, columns are work steps.

SolutionReset – The count of the number of solutions since a new best solution was found.

StartPositions – The matrix of all feasible starting points determined from *StartTest* matrix.

StartTest – The matrix of lowest values gathered per row from the sequence data matrix, used to determine feasible starting point.

StepList – The data point list input by the user in matrix format, contains the list of all process steps.

Storage – The storage location for all input data prior to solution generation.

Takt – The takt time which is input by the user.

Tau – The rate at which pheromones are deposited, input by the user.

TauEv – The rate at which pheromones evaporate, input by the user.

test – Used to separate the process for first Monte Carlo table entry.

TVGTime – The TVG time list input by the user in matrix format, times for all process steps are in this matrix.

WorkPackage – A matrix showing the work packages which have been generated with the current solution. Columns indicate work packages, rows work steps.

WorkPackageTime – A value used to calculate the amount of time already added to a work package during package splitting. Compared to takt time to determine when a new package must begin.

X – The number of steps in the problem.

13.10 Appendix J: Final Solution Process Descriptions

Table 24: Work Package 1 Process Description

Network Diagram Group	Group Description	Sub-Process Description
D	ACC Sub-assembly	Sub-assemble ACC to Bracket
K	Sub-assemble V-bar to Air Guide	Sub-assemble V-bar to Air Guide
G	Fit Dust Cap to Lock Support	Fit Dust Cap to Lock Support
A	Sub-Assembly 1 Preparation	Scanning
B	Sub-Assembly 1	Lower Defobox into Jig
		Upper Defobox into Jig
		Aligning Defoboxes
		Tightening Defoboxes
		Fit Lower Beam to Defobox
		Fit Upper Beam to Defobox
		Release Defoboxes and Carry to Carrier
		Lock Carrier
M	Fit Kerb Dampers	Fit Kerb Dampers

Table 25: Work Package 2 Process Description

Network Diagram Group	Group Description	Sub-Process Description
F	Fit Foam Absorber	Fit Foam Absorber to Upper Beam
C	Lower Air Guide	Clip Lower Air Guide to Defoboxes
O	Fit Brake Air Guides	Fit Brake Air Guides
E	ACC Fitment	Fit ACC to Frontend
H	Sub-Assembly 2	Fit Lock Support Assembly into Jig
		Fit Crossmember into Jig
		Fit Locks into Jig
		Pre-tighten Locks
		Assemble Upper Air Guide
		Pre-tighten Upper Air Guide to Cross Connection
		Lock Jig for Tightening
		Tighten Locks
		Unlock Jig
		Carry Assembly to Carrier
		Place Assembly on Carrier
		Lock Fixture on Carrier
J	Fit and Tighten US Crash Sensors	Fit and Tighten US Crash Sensors
S	Tighten V-bar Sides	Tighten V-bar Sides

Table 26: Work Package 3 Process Description

Network Diagram Group	Group Description	Sub-Process Description
P	Fit Upper Moek Bracket	Fit Upper Moek Bracket
X	Fit Front Bumper to Frontend (Ass. 1)	Carry Front Bumper to Carrier (Ass. 1)
		Bumper Inspection (Ass. 1)
		Assemble Front bumper (Ass.1)
AD	Collect and Load Rear Bumper (Ass. 2)	Collect and Load Rear Bumper (Ass. 2)
		Inspect Rear Bumper (Ass. 2)
AE	Fit C-Clips to Upper Air Guide	Fit C-Clips to Upper Air Guide
AK	Fit and Route Lock Bowden Cable	Fit and Route Lock Bowden Cable
W	Fit Bumper Support Bracket	Fit Bumper Support Bracket
		Insert Clip to Crossmember
N	Fit Horns	Fit Horns
L	Tighten Lock Support to Defobox	Tighten Lock Support to Defobox

Table 27: Work Package 4 Process Description

Network Diagram Group	Group Description	Sub-Process Description
T	Fit Snorkel	Fit Air Guide RHS
		Fit Snorkel RHS
		Fit Grill/Cap RHS
		Fit Air Guide LHS
		Fit Snorkel LHS
		Fit Grill/Cap LHS
		Scanning
I	Pre-fit Headlight Brackets	Fit Headlight Brackets
U	Place Tape on Headlights	Place Tape on Headlights
V	Position Headlights	Position Headlights
AF	Pre-Tighten Lights	Pre-Tighten Lights
AG	Scan Headlights	Scan Headlights
AH	Front End Alignment with Fixture	Front End Alignment with Fixture
		Tighten Upper Air Guide to Cross Connection
		Tighten Headlights
		Release Fixture

Table 28: Work Package 5 Process Description

Network Diagram Group	Group Description	Sub-Process Description
Q	Fit Lower Moek Bracket	Fit Lower Moek Bracket
R	Fit Moek Air Guides	Fit Moek Air Guides
AI	Clip Upper Air Guide Flaps to Bumper Beam	Clip Upper Air Guide Flaps to Bumper Beam
Y	Fit Front Bumper to Frontend (Ass. 2)	Carry Front Bumper to Carrier (Ass. 2)
		Bumper Inspection (Ass. 2)
		Assemble Front bumper (Ass.2)
AB	Fit Emblem	Fit Emblem
AA	Pre-tighten Front Bumper to Front End	Pre-tighten Front Bumper to Front End
AL	Apply Tape to Bumper and Headlights	Apply Tape to Bumper and Headlights
Z	Connect Harness to ACC Sensor	Connect Harness to ACC Sensor
AJ	Clip Bumper Z-Support Clips to Frontend	Clip Bumper Z-Support Clips to Frontend LH, RH and Centre
AM	Remove Frontend from Carrier onto LAM	Remove Frontend from Carrier onto LAM
AC	Collect and Load Rear Bumper (Ass. 1)	Collect and Load Rear Bumper (Ass. 1)
		Inspect Rear Bumper (Ass. 1)