8-2017

Balancing and Sequencing of Mixed Model Assembly Lines

Anas Alsayed Alghazi
Clemson University, aalghazi@gmail.com

BALANCING AND SEQUENCING OF MIXED MODEL ASSEMBLY LINES

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Industrial Engineering

---

by
Anas Alsayed Alghazi
August 2017

Accepted by:
Dr. Mary E. Kurz, Committee Chair
Dr. David Neyens
Dr. Kevin Taaffe
Dr. Joshua Summers

ABSTRACT

Assembly lines are cost efficient production systems that mass produce identical products. Due to customer demand, manufacturers use mixed model assembly lines to produce customized products that are not identical. To stay efficient, management decisions for the line such as number of workers and assembly task assignment to stations need to be optimized to increase throughput and decrease cost. In each station, the work to be done depends on the exact product configuration, and is not consistent across all products.

In this dissertation, a mixed model line balancing integer program (IP) that considers parallel workers, zoning, task assignment, and ergonomic constraints with the objective of minimizing the number of workers is proposed. Upon observing the limitation of the IP, a Constraint Programming (CP) model that is based on CPLEX CP Optimizer is developed to solve larger assembly line balancing problems. Data from an automotive OEM are used to assess the performance of both the MIP and CP models. Using the OEM data, we show that the CP model outperforms the IP model for bigger problems. A sensitivity analysis is done to assess the cost of enforcing some of the constraint on the computation complexity and the amount of violations to these constraints once they are disabled. Results show that some of the constraints are helpful in reducing the computation time. Specifically, the assignment constraints in which decision variables are fixed or bounded result in a smaller search space. Finally, since the line balance for mixed model is based on task duration averages, we propose a mixed model sequencing model that

minimize the number of overload situation that might occur due to variability in tasks times by providing an optimal production sequence. We consider the skip-policy to manage overload situations and allow interactions between stations via workers swimming. An IP model formulation is proposed and a GRASP solution heuristic is developed to solve the problem. Data from the literature are used to assess the performance of the developed heuristic and to show the benefit of swimming in reducing work overload situations.

DEDICATION

To my parents for their support, encouragement and prayers.
To my wife for her care, endurance, and endless love.
To my children for being the joy of my life.

ACKNOWLEDGMENTS

First and foremost, all praise and thanks to Allah the most merciful and the most gracious for his guidance and blessings he bestowed upon me and my family.

I would like to express my deepest gratitude to my dissertation advisor, Dr. Mary Beth Kurz for her guidance and support through this journey. I am honored to have her as my advisor, her support in my research is unmatched by anything else. The dedication and attention she pays to her students and their work is amazing. She will always be one of my great role models. I would like also to show my gratitude to my committee members Dr. Joshua Summers, Dr. Kevin Taaffe, and Dr. David Neyens for their valuable advice and comments. I am also grateful for the valuable comments and support that our Department Chair Dr. Cole Smith has provided to me in this journey.

Special thanks are extended to the faculty members and staff in the Department of Industrial Engineering. I am thankful for Martin Clark and Brandon March for the technical support they provided to me. Words of appreciation are due to my colleagues in the Department of Industrial Engineering and to my friends. Special thanks to Bryan Pearce for the fruitful discussions we used to have about the line balancing problem. Thanks to him and to the team at BMW for effort they made in collecting data.

I am thankful to my parents Alsayed Alghazi and Afaf Bahjat for their guidance and prayers, my brothers and sisters for their unconditional love and support. I am thankful to my lovely wife and friend Afaf Effat who has been always on my side during times of serenity and stress. Without her love and support this work would have been much harder.

I cannot thank her enough for putting up with me during my studies. Finally, I am thankful to my children Albatool, Omar, Sofana, and Suhyla for being the gifts that always reminded me what is important.

TABLE OF CONTENTS

Table of Contents (Continued)

Page

LIST OF TABLES

List of Tables (Continued)

Table Page

LIST OF FIGURES

List of Figures (Continued)

Figure Page

CHAPTER ONE

1 INTRODUCTION

1.1 Assembly line balancing problem

An assembly line is a flow-oriented production system in which final products are assembled from components while flowing through a sequence of serially aligned production units known as stations. Production is initiated with a job which is a workpiece that is launched down the line. This workpiece flows through stations using a transportation system such as a conveyor belt. At each station a number of operations, known as tasks, are performed on the workpiece. Each task requires a certain amount of time to be performed in addition to other requirements like tools. Due to technological constraints or the physical structure of the product, there are relations between tasks that necessitate finishing some tasks before starting others. These constraints are captured in a precedence graph in which each task is represented by a node and each precedence relation is symbolized by a directed arc.

The assembly line balancing problem (ALBP) is the decision problem of finding a feasible line balance; in other words, assigning each task to one station such that the precedence constraints and any additional restrictions are fulfilled. For the case of paced assembly line, the cycle time determines the maximum time a workpiece can spend at each station. The line balance is feasible only if the total sum of task durations at any given station, named the station load, does not exceed the cycle time. In addition, if the station load is less than the cycle time then the idle time for each cycle would be the difference between the station load and the cycle time. The word “balancing” stemmed from the fact

that minimizing the total idle time across all station would yield a balanced workload across all stations which can be achieved by minimizing the number of workstations. This problem was named the simple assembly line balancing problem (SALBP) by Baybars (1986). The assumptions for the SALBP are: production of one product, paced line with fixed cycle time, deterministic operation times, no assignment restrictions other than the precedence constraints, serial line layout with one sided stations, and identical stations in terms of machines and workers.

1.1.1 Classification

Different objectives can be used in the decision problem of assembly line balancing; the objective used should reflect the strategic goals of the organization. While using a cost/profit type of objective is tempting, it may not be the best choice because of the errors that may occur when estimating either the cost of operating the line over a long period of time or the profit from selling the final products. For this reason, other objectives are used which define four versions of the SALBP. The first version utilizes the objective of minimizing the number of stations for a given fixed cycle time which is referred to as SALBP-1. The second version utilizes the objective of minimizing the cycle time for a given fixed number of stations which is named SALBP-2. The third version is SALBP-E which maximizes the line efficiency, and the fourth version is SALBP-F which searches for a feasible solution given a fixed number of station and cycle time. (Scholl, 1999)

Traditionally the research in the assembly line balancing field was focused on the SALBP with all of its restricting assumptions. However, recently a lot of research is generalized because real life line balancing problems require more than the set of limiting

assumptions of the SALBP. In the simple problem, other practical consideration are overlooked in the model such as having more than one product, parallel work stations and tasks, zoning restrictions, processing alternatives, stochastic processing times and U shaped assembly lines. According to Baybars (1986), the attempts to extend the SALBP by integrating practical issues to make the problem more realistic are categorized under the general assembly line balancing problem (GALBP). In order to structure the research in the field of ALBP in an attempt to close the gap between academic discussion and practical applications, Boysen et al. (2007) introduced a tuple notation similar to the one adopted in the machine scheduling field so that any ALBP can be easily characterized by it. This tuple notation characterizes the problem according to: precedence graph characteristics, station and line characteristics, and objectives. Using this notation more than half a billion variations of the ALBP can now be easily characterized. This notation also accommodates more realistic ALBP after the structure of SALBP have been well studied and more research is going to be geared towards the applicable realistic line balancing problems.

1.1.2 Mixed Model Assembly Lines

Assembly lines in which one product is assembled are categorized as single model. If more than one model or version of the product is assembled on the same line, the assembly line is categorized either as mixed-model or multi-model depending on how these different models are intermixed on the assembly line. For random inter-mixing of models the problem of balancing the line is referred to as the mixed model assembly line balancing problem (MALBP). The different line types are shown in Figure 1 in which different models are represented by different geometric shapes. Usually models may differ from

each other in size, color, material or any other specifications such that their production may require new tasks, different task times, or different precedence constraints. However, most of the time all models contain identical (or interchangeable) parts and usually their production process is similar. The difference could be as simple as options that are either present or absent from the desired model. (Becker & Scholl, 2006)

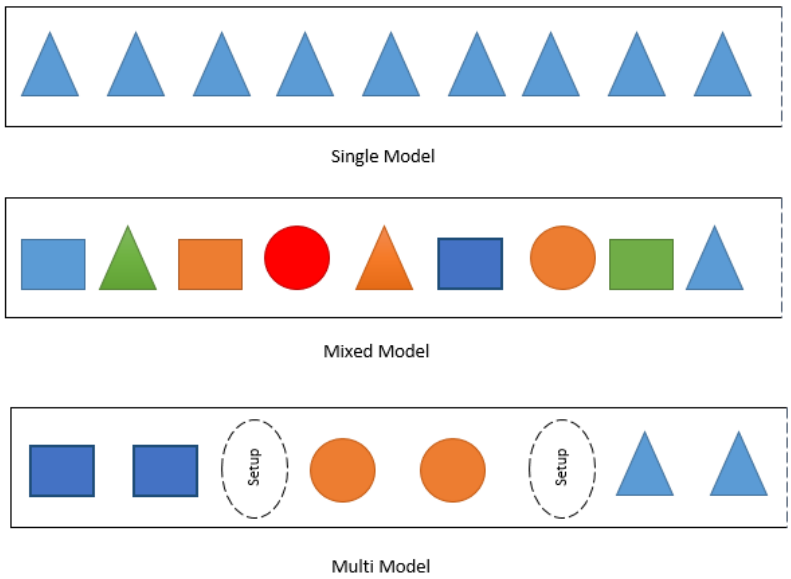

Figure 1: Types of assembly lines

1.1.3 Mixed Model Line Balancing and Sequencing

In managing the mixed-model assembly line, several decision problems with different planning horizons need to be solved. For the long term horizon, the decision on the assignment of tasks to workstations (line balancing) needs to be taken. This includes other decisions like determining the number of workstations, station workload, production rate or equivalently, the cycle time. All of these decisions are part of the MALBP. As in the single model problem, there are different versions of the MALBP depending on the

objective sought. However, the mixed model problem is much harder than the single model as station utilizations' need to be balanced with respect to each model or even across all models. Thus, single model objectives are not adequate in this case. In addition, the cycle time for each of the mixed models is different from the one used in the single model case and it is not considered as an upper bound to the station time because average task durations are used for the mixed model case. So if there are some models with station times that are significantly smaller than the cycle line, it is expected that for some other models the station time will exceed the cycle time. This in turn will lead to stations with idle time or work overload depending on the sequence of models production.

For mixed model line management, the long term decision of determining the line balance and the desired production rate is not enough. This is due to the inefficiencies that result from the variation of station utilizations with respects to the models, which is a result of using averages of task durations across all models to come up with line balance. To deal with this, a short term decision problem is used to find a sequence of models that satisfies the demand over the production period and optimize some given objective. This problem is named the mixed model sequencing problem (MMS). This problem is considered short term since it arises every production shift which could be daily or weekly. The balancing and sequencing problems are interrelated and each depends on the other. The actual performance of a line balance that is achieved by using task averages depends on the order of the production sequence. If models with high task times are consecutive in a production sequence some stations will require more than the cycle time to finish the assembly tasks assigned to it. This will cause a work overload that disturbs the assembly line and might

cause it to stop incurring costs. One way to avoid work overload is to alternate high and low demanding models such that work overload is minimized. Mixed model sequencing takes into account worker movement in stations and uses different model task duration to determine the best production sequence that minimizes the work overload.

1.2 Constraint programming background

Operations Research (OR) has been the traditional field of research for solving combinatorial problems like scheduling. In OR, solutions are achieved by solving simple mathematical models that heavily exploit the combinatorial structure of the scheduling problem to get the best performance. So "OR" is more about achieving a high level of efficiency by using the proper solution algorithm. However, simplifying the problem by imposing non-practical assumptions and removing constraints is unavoidable in order to solve a practical combinatorial problem like scheduling using the classical OR models. In some cases, this makes the solutions to these simplified models uninteresting because they are not applicable as a solution to the real problem. On the other hand, artificial intelligence (AI) research in scheduling tends to explore more general scheduling models and solution paradigms. AI algorithms focus on providing general algorithms to solve a wide range of problems. Because of the emphasis on general algorithms and when compared to OR algorithms, AI algorithms may perform poorly on some problems. Constraint programming was introduced to combine the benefits of OR and AI such that we have algorithms that are both efficient and can be applied to a wide variety of problems (Baptiste et al. 1995).

Constraint programming (CP) is a declarative programming paradigm that was initially influenced by computer science, or specifically artificial intelligence (AI), and programming languages. However, recently operations research (OR) has been a major influence to CP. The interface and modeling aspect of CP is close to the computer science field but recently the solution strategies and efficient algorithms in CP has been influenced by OR. Generally, CP is used to solve what is called the constraint satisfaction problem

(CSP). An instance of the CSP is composed of a set of variables, a domain for each variable that defines the values to which the variable may be assigned, and a set of constraints that define the relation between the values of one or several variables. Solving a CSP involves assigning values to variables such that all constraints are satisfied. If a feasible solution is found by assigning values from the domain to each variable, the problem is said to be satisfiable. If no assignment of values to variables from their domains that satisfy all constraints, the problem is said to be unsatisfiable.

Sometimes it is not useful to only get a feasible solution; instead we want to find a solution that is optimal with respect to a certain criterion. This problem is called the constraint optimization problem (COP) which is simply a CSP with an objective function. CSP can be modified to solve COP by creating an objective variable that represents the objective function. When an initial feasible solution is found, a new objective constraint is added that forces the (new) objective variable to be better than the initial solution found. This is repeated every time a new better feasible solution is found until the problem becomes unsatisfiable in which the last feasible solution found is the optimal solution to the COP. It should be noted that there are different approaches to tackle the CSP. One approach is using IP techniques such as the cutting plane method, and the Branch and Bound algorithm (B&B). Another approach is by using metaheuristics such as Simulated Annealing (SA), Tabu Search (TS), and Genetic Algorithms (GA). The term constraint programming in general means a computer implementation to solve the CSP but it has been also used in the literature to denote implementing these algorithms in a conventional logic programming language. Enhancements to logic programming languages have been made

to overcome some of the insufficiencies and new languages have been developed such as Constraint Handling in PROLOG (CHIP). During this time, some authors started using the term constraint logic programming (CLP) in the literature in place of CP which might be confusing. Another approach to tackle CSP was implementing CP using general purpose programming languages like C++ or a special declarative language such as the OPL language. Several commercial packages provide a graphical interface environment for CP such as IBM ILOG CPLEX that is based on the OPL language or the Fico Express optimization suite which is based on the Mosel language. Recently, algebraic modeling languages such as AMPL and AIMMS have been extended to allow CP. There are also many free CP modeling languages that can be considered medium level modelling languages that usually require a low-level CP solver. The area of CSP research has received more attention lately with the advancement in computer hardware which has made efficient CP algorithms possible (Brailsford et al. 1999).

CP variables may have different types of domains, such as integer, logical, or any non-numerical domain. Also, unlike linear programming, different types of constraints can be used for each type of variable such as arithmetic and logical constraints. The solution procedure for CP involves a combination of domain reduction, constraint propagation, and search. To demonstrate these techniques, a CSP can be presented as a graph in which the nodes are variables represented by their domain and the arcs are the constraints between these variables. We use examples from Fromherz (2001) to demonstrate domain reduction and constraint propagation techniques. Domain reduction is the direct application of constraints on a variable's domain. For example if integer variable X has a domain [0,10]

and there is a direct constraint X>3, the domain of X is reduced to [4,10]. If we have another integer variable Y with the domain [0,10] that is connected to X via a constraint X<Y-3, then the changes in the domain of X are propagated to change the domain of Y to be [7,10]. The change in Y's domain is propagated again to change the domain of X to [4,7]. This process is called domain propagation and results in what is called arc consistency in which inconsistent values are removed from the variables' domains along the arcs of the graph. The process of constraint propagation and arc consistency are illustrated in Figure 2.

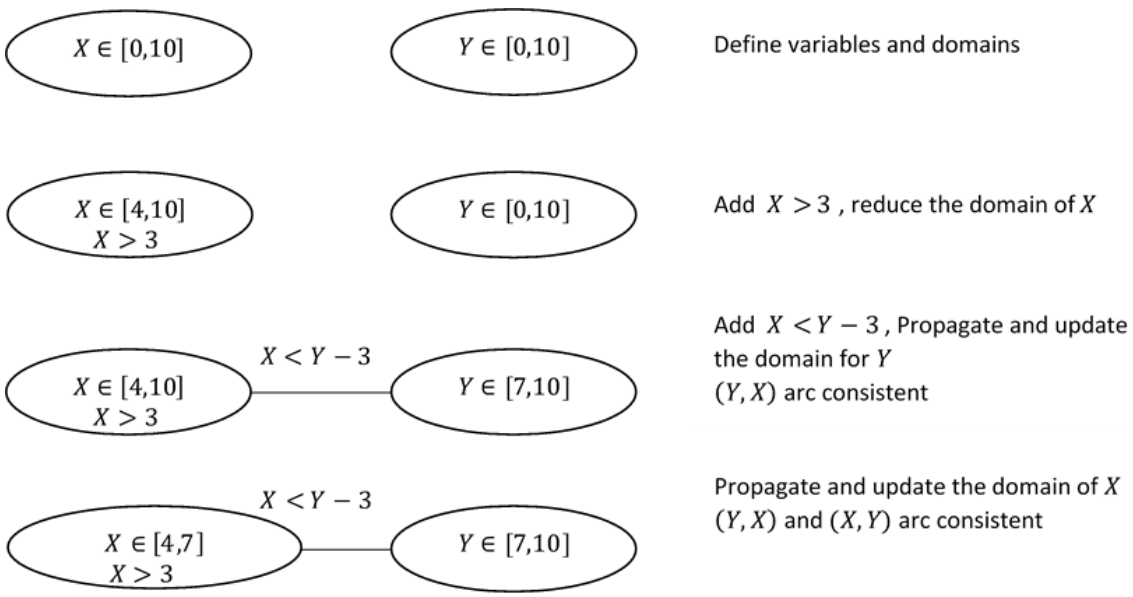

Figure 2: Propagation and arc consistency

The main aim of constraint propagation and domain reduction is to remove values from variable's domains that will certainly lead to infeasible solutions in hopes to reduce the search space. Usually constraint propagation is denoted as an incomplete procedure as not all inconsistent values are removed from the variables' domains. If all domains are reduced to one variable then a feasible solution might be found as a result of constraint

propagation only but a feasible solution is not guaranteed. In most cases search is usually needed to supplement constraint propagation. The search procedure enumerates assignments of values to variables, when a variable is fixed during search constraint propagation is used to reduce the domains of variables that are not fixed in order to reduce the search space.

The search in CP is usually done as a depth-first tree search with domain filtering. It is a constructive algorithm meaning that each variable is assigned a value incrementally until a feasible solution is found or a constraint is violated which implies infeasibility. If a constraint is violated the last variable assignment is undone and the variable is assigned another value from its domain. If there is still no feasible solution backtracking occurs in which the search backtracks to a previously assigned variable. The search continues until all branches in the search tree are explored. The search efficiency is affected by the way the search tree is built in terms of variable ordering; also value ordering is an important factor. There are several heuristics that can be used to choose a better variable and value ordering in order to minimize backtracking and increase the search efficiency. In addition, there are special heuristics that can be used to exploit the structure of some problems and in turn increase the search efficiency by improving domain filtering which is the process of removing variables values that will not be part of any feasible solution. Specialized algorithms are developed for different constraints to help with domain filtering. Also, evolutionary algorithms, dominance rules, large neighborhood search, and machine learning are examples of techniques that are incorporated in CP search mechanism.

1.3 Integer programming and constraint programming

Research in the OR field developed a mathematical programming framework with tools such as linear programming (LP) and integer programming (IP) to solve combinatorial optimization problems, while research in the AI field and specifically constraint satisfaction and logic programming along with some of OR ideas developed the CP framework. What is common between IP and CP is that their solution method is based on a tree search in which branching is used to explore the search space. The main difference between the two is how the tree is explored and how the branching decisions are made at each node of the search tree. For example, in IP the branching is done on a node using LP sub-problems that are generated by relaxing the integrality constraints. The node is fathomed in case the LP sub-problem has no feasible solution, or has an integer solution that is better than the best solution found, or has a solution that is worse than the best solution found. If the solution to the sub-problem is better than the best solution found yet does not satisfy the integrality constraint a new branching step is required. This process is repeated until the search terminates when it is proven that there is no feasible solution available that is better than the best solution found. On the other hand, CP uses domain reduction and constraint propagation at each node to reduce the domain of each variable. The node is fathomed if domain reduction reduced all domains such that they are empty, and the search continues to enumerate all remaining possible solutions after further domain reduction and constraint propagation steps are done. The search terminates when all nodes have been fathomed. It should be noted that the constraint propagation algorithm used is a key factor in the efficiency of the CP (Jain & Grossmann 2001).

Hooker (2002) compared optimization to CP and gave several examples on the differences. The first is the word “programming” in CP which refers to computer programming since the problem is usually formulated using a declarative modeling language. Usually the modeling language “programming” in CP gives the user great control over the search procedure using a lower level language like Java or C++. However the word “programming” in linear programming is not related to computer programming as the name originated when George Dantzig developed linear programming formulations to solve planning problems (or programs) in the US Air Force back in 1946. Another distinction is the use of inference in both CP and IP. In CP inference is done to reduce the search space directly using domain reduction and constraint propagation. On the other hand, cutting planes in IP might be considered an inference tool that creates better relaxations and speed up the search indirectly.

1.4 Contribution and research questions

Much of the research done in the line balancing problem literature does not consider most of the real life restrictions that are facing decision makers managing assembly lines. In well-known car manufacturing plants, assembly line balancing is still done manually over several days since no applicable tool is available for them. The GALBP started to appear in the literature in which the limited assumptions of the SALBP are relaxed so that it becomes more applicable in real life instead of being just in academic research papers. However, research in the GALBP usually concentrates on one generalization and extends the model in that direction only. In this research, the MALBP with parallel stations, zoning constraints, and assignment restriction is extended by adding more detailed task assignment constraints, worker interference constraints, and ergonomics. For task assignment constraints, instead of having two constraints for station/worker incompatibility, the extended model has four constraints for tasks that need to be done by either the same worker, or different workers and tasks the need to be done in the same station, or in different stations. In the extended model, tasks are assigned to workers instead of fixed workplaces that are tied to certain mounting positions. This way each worker can be assigned any task regardless of the mounting position. To avoid worker interference, a new constraint is added to avoid the overlap of any two tasks that share the same mounting position in any station. Furthermore, unproductive movement is avoided by limiting each worker from being assigned tasks with opposite mounting positions. Finally, ergonomics are added to the extended MALBP model so that worker health is not endangered by assigning him too many ergonomically demanding tasks that might lead to a lifelong injury.

In this research, an IP is presented for our extended MALBP model with parallel stations, zoning constraints, assignment restrictions, and ergonomics. A CP model is proposed to solve the problem using the scheduling module of the CPLEX CP optimizer. To the best of our knowledge, there is no published research that uses CP to solve the MALBP with parallel stations, zoning constraints, assignment restrictions, and ergonomics.

Subsequently, an extended sensitivity analysis is done using data from an automotive OEM to measure the cost of adding the different constraints on the processing time needed to reach the optimal solution. The problems' constraints are divided into hard constraints that make up the assembly line balancing problem, and soft constraints that can be relaxed without violating the basic assumptions of the line balancing problem. Experiments are done to build intuition about the relative cost of modeling and forcing some of the realistic constraints in the mixed model line balancing problem.

Because the mixed model line balancing is done based on task average times, the related problem of MMS is studied to minimize any disturbance to the assembly line that might be caused by work overload in any station. An IP is proposed with the objective of minimizing the work overload situations. The assumption for the proposed MMS model is based on the skip policy in which the utility worker works exclusively on any work piece whenever an overload situation is foreseeable while the normal worker skips this work piece to the next one in sequence. This problem is discussed in the literature with assumption of closed stations. That is no worker can swim past his station's boundaries to

complete a task if needed. In this research, we extend the work done by assuming open stations and develop a heuristic to solve the problem.

The research questions addressed in this dissertation are as follows:

1- Can we model the mixed model line balancing problem with parallel stations, zoning, assignment, and ergonomic constraints as a scheduling CP? If so how efficient is the CP model when compared to the IP in terms of computation time?
2- How sensitive are the IP and CP models' computation time to the maximum ergonomic score limit? What is the impact of removing some of the constraints on the solution quality and computation complexity?
3- Is there a more efficient way to solve the mixed model sequencing problem? What is the benefit of considering workers swimming?

1.5 Dissertation organization

This dissertation is organized as follows. Chapter 1 introduces the dissertation by defining the assembly line balancing problem and presenting the different classification of the problem. This is followed by introducing the mixed model assembly lines and how are they different from the single and multi-model assembly lines. Then, the interrelated problems of mixed model balancing and sequencing are defined. Constraint programming background and its relation to integer programming is given next. Chapter 1 concludes with research contribution and organization.

Chapter 2 presents the problem of mixed model line balancing with parallel stations, zoning constraints, and ergonomics. The problem description is given first using an example. Then a literature review on related work is presented followed by the

motivation for this research. The mathematical model along with the integer program is presented followed by the proposed constraint programming model. The performance of the proposed models is shown in the computation experiments section along with discussion of the results.

In Chapter 3, a case study is presented in which sensitivity analysis is done on both the proposed models to assess the cost of enforcing some constraints on the computation complexity along with the amount of violations occurred when a constraint is disabled.

Chapter 4 introduces the problem of mixed model sequencing and how is it different from the car sequencing problem. This is followed by a review in related work. Next, problem description is given and illustrated by an example. Model formulation is presented in the next section followed by the mathematical integer programming model.

CHAPTER TWO

2 MIXED MODEL LINE BALANCING WITH PARALLEL STATIONS, ZONING CONSTRAINTS, AND ERGONOMICS

2.1 Introduction

An assembly line is a flow-oriented production system that is composed of productive areas called stations arranged in a serial manner. The final product starts as a work piece that is launched down the line passing by all stations on some kind of transportation system such as a conveyor belt. At each station, a number of assembly operations are performed on the work piece. The amount of time each work piece is within the boundaries of a station depends on the conveyor speed, which is also defined by the interval between launching work pieces, is called the cycle time. Generally, the total amount of assembly operations done on the work piece by a single operator at any given station cannot exceed this cycle time. In addition, the assembly operations required to be done on the work piece have some precedence constraints because of technological and organizational conditions. The decision problem of assigning the different assembly operations to stations with respect to some objective and satisfying the cycle time limitation and the precedence requirement between operations is referred to as the assembly line balancing problem.

Initially, assembly lines were used as a cost efficient way for mass production of identical products. However, in order to respond to customer needs, companies added the option to customize the products they offer. Automation and multipurpose machines made it easier to produce customized products on the same line in a cost effective way similar to

the mass production counterparts. Yet these assembly lines are associated with considerable investment costs, which makes the configuration of these lines an important factor to still make it cost efficient. The configuration of the line includes management decisions that are related to setting system capacity, such as the cycle time, as well as task assignment to stations.

Most of the research in the configuration planning of assembly systems is focused on the problem of line balancing. Since most of the work in the literature is based on simplifying assumptions, the research was named simple assembly line balancing problem (SALBP) in the review paper by Baybars (1986). Some later studies extended the simple problem by incorporating practical aspects such as parallel stations and zoning constraints; this type of research is categorized under the general assembly line balancing name in the survey by Becker and Scholl (2006). Despite this effort in extending the problem to be more realistic, there is still a gap between academic papers and practical implementation. To this day, many automotive assembly organizations still line balance manually. Boysen et al. (2007) tried to explain the reasons behind this gap between the academia and practice. The first reason is that researchers have not considered the true real world problem yet. The second reason is that the problem was studied but no satisfactory solution was found to it. Finally, the third reason is that scientific results could not be translated into something practical. The classification scheme proposed in Boysen et al. (2007) was the first step to close this gap between academia and real life.

The assembly line balancing problem is not just the long term decision of efficiently designing the assembly line by determining the number of stations required, as re-balancing

is needed periodically when production processes, overall demand or demand mix changes. Choosing the optimization objective impacts the resulting solution and depends on the strategic goals of the company. Instead of using a cost/profit type of an objective, a surrogate objective is often used to maximize the line utilization or minimize the number of workers needed.

Becker and Scholl (2009) consider the problem of balancing assembly lines with variable parallel workplaces in which the objective is to minimize the number of workplaces needed. A workplace is the combination of a worker at a station, in the case of multiple workers per station. The model proposed is based on the simple assembly line assumption of mass production of a homogenous product. In this work, we extend the model of Becker and Scholl (2009) to the mixed model environment and account for ergonomic constraints, usually ignored in line balancing models. We develop both an integer programming (IP) model and a constraint programming (CP) model for the problem at hand and assess the performance of both models using data from industry.

In this chapter, the problem description is given first with details about the different constraints that define the problem at hand. This is followed by an example to illustrate the different constraints of the problem and the solution structure in section 2.2. Next, a literature review on related work is given in section 2.3 followed by the motivation behind this work in section 2.4. In section 2.5 the mathematical program is given followed by the constraint programming model in section 2.6. Computational experiments and results discussion are presented in section 2.7. Finally, concluding remarks and future possible research directions are given in section 2.8.

2.2 Problem description

We describe the decision problem that is intended to solve the realistic mixed model line balancing problem by an example. To achieve realism, we incorporate several conditions and limitations that are faced in an industry's real assembly line. The constraints that make a line balancing problem applicable in real world industrial setting are explained in detail in Falkenauer (2005). Our problem is based on the assembly line balancing problem with flexible parallel workstations that was introduced by Becker and Scholl (2009) and an OEM's environment in which the objective is to minimize the total number of workers on the line subject to several constraints.

The characteristics of the problem that are similar to the characteristics of the SALBP are: serial line layout, fixed cycle time (paced line), and deterministic task times that are less than the cycle time. In addition, to capture realism the following characteristics are included: mixed model assembly, parallel workers, zoning constraints, assignment restrictions and ergonomic risk constraints.

2.2.1 Mixed model environment

In a mixed model environment, different base models known as variants that may include a variety of options are assembled on the same line. Since some automakers provide their customers with the flexibility in customizing the cars they order with the options they want, these automakers usually operate under a make-to-order environment. The number of options/variants combinations lead to a huge number of possible customized cars to be assembled on the same line. The installation of different options typically leads to

variations in assembly task times; for example the installation of a power liftgate requires a different amount of time when compared to the manual liftgate installation.

There are two primary approaches to deal with variation in task durations when it comes to enforcing the cycle time constraint. The first is enforcing the cycle time on the average task times, that is the sum of average task times assigned to a worker should not exceed the cycle time. The drawback of this method is having some models exceed the cycle time at some stations. The second method is enforcing the cycle time on all model variations. The drawback of this method is that it requires a higher cycle time which is translated into a lower production rate. Boysen et al. (2008) reviews the different approaches used to handle the mixed model problem in the literature. In this chapter, the first approach is used in which the average task time is guaranteed to be less than the cycle time. In addition, because of the huge number of models that could be configured, automakers use option-based forecasts instead of figuring out the forecasts for each possible model. The option-based precedence graph introduced by Boysen et al. (2009) will be used in this research based on the proportion p_h of products (cars) that require task h with duration d_h . This probability is derived for a production run using the forecasted demand of the volume of cars that will require the execution of that task divided by the total production volume. The weighted task time for task $h, \bar{d}_h$ is calculated as follows $\bar{d}_h = p_h d_h \quad \forall\, h.$

2.2.2 Zoning constraints

The zoning constraints are used to avoid worker interference in a station and reduce non-productive walking time between mounting positions. Each task to be executed by any

worker has to be done in a specific location on the product. This attribute is called the mounting position and it is one of the different positions available in a product zone map. Figure 3 shows an example of a product zone map in which there are nine different mounting positions.

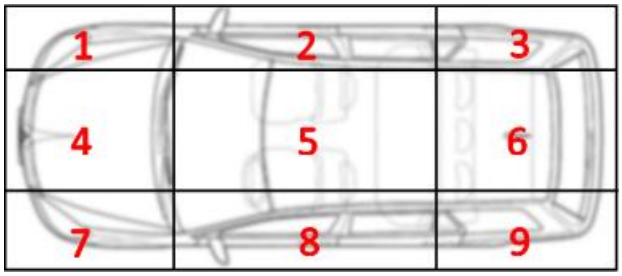

Figure 3: Product zone map

Each task h has a mounting position q_h from the 9 positions on the product zone in which the task takes place. Mapping each task to a mounting position helps in prohibiting the overlap in the scheduling of two tasks h, l with the same mounting position $q_h = q_l$ at any station. Figure 4 shows an example where interference might happen when two workers execute tasks that share the same mounting positions at the same time.

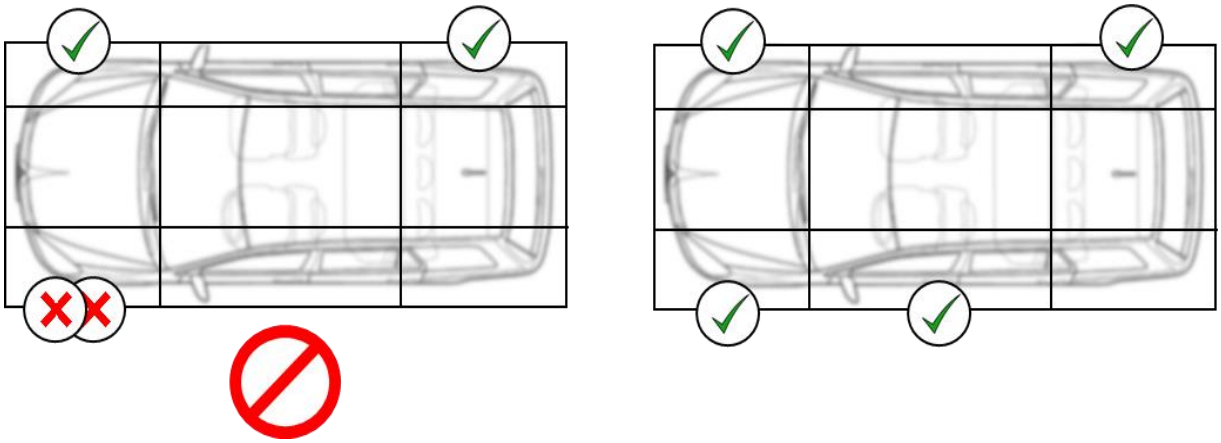

Figure 4: Worker interference example

2.2.3 Parallel workers

One operator per station is assumed in the simple assembly line balancing problem's literature. However, it is common to have more than one operator working within the same station in big product assembly lines such as cars, trucks, and big machines. More specifically in the automobile industry, the assembly of cars involves using relatively large parts and thus many stations have enough space to accommodate multiple workers who are able to work at the same time. We assume that each worker is not fixed to one work area inside the station and can be assigned different tasks that have different mounting positions. Furthermore, we provide detailed scheduling of the tasks assigned to each worker to avoid conflicts between workers in the same station and indicate if the line balance is actually feasible. Infeasibility could occur even if the sum of all tasks assigned within a station is less than the cycle time. This is because there exist precedence relations between tasks that could lead to idle time if one operator is waiting for another to finish a task.

2.2.4 Assignment restrictions

Some tasks are restricted to be assigned to some stations due to different reasons. For example, a task that requires a resource that is available only on some stations or a task that is done below the car and requires the car to be elevated. Different assignment restrictions are needed in order to have a flexible tool that can be useful for real life line balancing. These restrictions can be categorized as follows

2.2.4.1 Resource constraints

This constraint is used when a task requires a certain resource such as a tool that is not available in all stations. For example, some tools are heavy and expensive to move between stations so they will be fixed to certain stations. Tasks that require this tool should be only assigned to this station. For a task h that require a certain resource not available in all stations there is a set of eligible stations U_h that contains all stations this task can be assigned to. If a tool is available in only one station, tasks that need this tool are assigned to this station in the preprocessing step.

2.2.4.2 Adjacency constraints

This constraint is for tasks that need to be executed consecutively by the same worker. For example, a task for picking up a part must be followed by a task to assemble it. In this case, the worker is not open to any other tasks since he has the part in his hands. Adjacency constraints are treated in the preprocessing phase by combining the adjacent set of tasks into one task.

2.2.4.3 Same worker constraints

This constraint is for tasks that need to be executed by the same worker but does not need to be done consecutively. An example would be inspection or quality checks that should be done by a worker after finishing a task. Since the inspection may involve several tasks, the same worker constraint is used instead of the adjacency constraint. All tasks constrained to be done by the same worker are collected in sets labeled by R^{sw}.

2.2.4.4 Incompatible worker constraint

This constraint is needed when a set of tasks are not allowed to be assigned to the same worker. This constraint is mainly used to prohibit unproductive walking by limiting the assignment of tasks with opposite mounting positions to the same worker. An example for the product zone in Figure 3 would be to limit any worker from being assigned a pair of tasks h, l with mounting positions $\{(q_h, q_l)\} = \{(1,9), (7,3), (4,6)\}$. That is, if a worker is assigned a task with mounting position 1 the assignment of all tasks with mounting position 9 is prohibited for him since it would be unproductive to walk all the way from product zones 1 to 9. All tasks that are incompatible worker are collected in sets labeled by R^{nw}.

2.2.4.5 Same station constraint

This constraint is used to indicate that a set of tasks needs to be done on the same station but not necessarily done by the same worker. An example would be a large piece that needs to be installed by two workers via two different tasks. Both tasks need to be done on the same station so that each worker would be assigned one of them. All tasks that need to be assigned to the same station are collected in sets labeled by R^{ss}.

2.2.4.6 Not the same station

This constraint prohibits a set of tasks from being assigned to the same station. An example for using this constraint would be for a task to lubricate a movable part and task to install the seat. These two tasks should be assigned to different stations to avoid the danger of soiling the seat with the lubrication material. Another example would be a gluing

task that needs time for sitting before executing another follower task. All tasks that are station incompatible are collected in sets labeled by R^{ns}.

2.2.5 Ergonomics risks constraints

The ergonomic risks are treated in two different ways, directly and indirectly. An indirect way of treating ergonomic risk is divided to station-related and worker-related methods. An example for dealing with an indirect station related ergonomic risk is the relation between tasks and the position of the car on the line. Some tasks that require installing parts above the car need the car to be tilted to avoid injury for the worker executing it. These tasks should be restricted to station in which the car can be tilted. Furthermore, tackling the worker interference issue and unproductive walking are considered to be indirect worker level ergonomic risk constraints. On the other hand, the direct application of ergonomic risk mitigation can be accomplished through the use of a weighted ergonomic risk score $\bar{g}_h$ associated with each task h to make sure that each worker is assigned tasks with a total weighted ergonomic risk score less than a predefined maximum recommended level G.

2.2.6 Pre-processing

The pre-processing phase is important to prepare the input before it is used with the mathematical program. For the mixed model problem, this involves computing the weighted task durations and the weighted ergonomic scores. Furthermore, in order to reduce the complexity of the solution techniques, some of the available information is useful in lowering the number of decision variables by fixing some of them. Also, it is possible to remove unnecessary constraints by reducing the set of variables the constraint

applies to. This helps in getting a tighter feasible solution space and helps in reducing the combinatorial explosion that happened with these type of problems. The pre-processing phase involves the following:

- Reduce the number of tasks by combining the tasks in an adjacency set into one task.
- Defining the feasible stations set F for all tasks via the use project management techniques to calculate the early start and late finish for each task.
- Defining the eligible stations set U for all tasks that require a specific resource that is not available in all stations. If this resource is available in one station only, all tasks that require it are pre-assigned to this station.
- Calculating the weighted task duration $\bar{d}$ and the weighted ergonomic score $\bar{g}$ for all tasks.
- Populating the assignment constraints sets R^{sw}, R^{nw}, R^{ss}, and R^{ns}.

2.2.7 An example

Consider the example instance with 12 tasks and the option-based task precedence graph in Figure 5. Assume that 10 cars are to be assembled, each with different options that require different tasks. The volume and proportion of each task are given in Table 1 along with the task duration and ergonomic score. The weighted values of both the task duration and the ergonomic score are calculated based on the volume of cars that require that task. It should be noted that both the ergonomic and cycle time constraints are enforced on the weighted (average) values only; that is the total weighted task duration and ergonomic

scores are less than the cycle time and the ergonomic risk maximum recommended level respectively. Figure 6 shows the mounting position for each task, which is the location in which the task will take place on the car. Since the number of stations is predefined, the early start and late finish for each task are calculated and shown in the Gantt chart (Figure 7) with each task scheduled at its early start time. The Gantt chart helps reduce the feasible station set for each task, for example tasks 1, 6, and 12 can only be assigned to stations 1, 2, and 3 respectively. The assignment constraints are given in Table 2. Since tasks 7a and 7b are in an adjacency set, they are joined together as task 7 in the preprocessing phase. The "incompatible worker" assignment constraint is used in this example to avoid unproductive walking between opposite mounting positions. For example a worker who is assigned a task with mounting position 1 will not be assigned another task with mounting position 4 and vice versa but he can be assigned tasks with mounting positions 2 and 3. An optimal solution to this example is shown in Figure 8 in which all 6 potential workers are assigned tasks. In Figure 8, the tasks are represented by blocks in which the length denotes the weighted duration of the task and the height of the block denotes the weighted ergonomic score. Since task 6 has a high weighted ergonomic score it is the only scheduled task for worker (2,1). The scheduling of tasks to workers is important for reasons other than avoiding worker interference since just assigning tasks to workers might yield an infeasible solution because of the precedence relations. For example worker (1,1) has an unavoidable idle gap since he had to wait for task 1 to finish before starting task 5 because of the precedence relation between the two tasks.

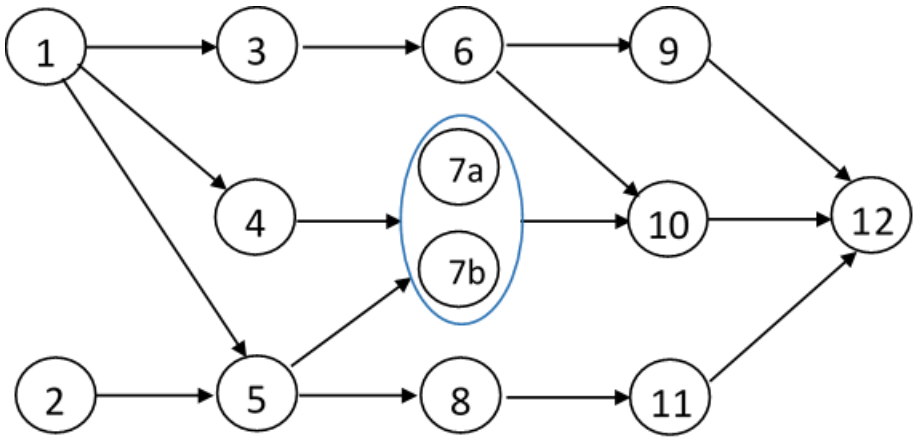

Figure 5: Option-based joint precedence graph

Table 1: Example data for the line balancing problem

Task	Volume	Proportion	Mounting position	Duration	ErgoS	Weighted Duration	Weighted ErgoS
1	10	1	1	50	50	50	50
2	10	1	2	30	40	30	40
3	3	0.3	3	80	70	24	21
4	5	0.5	3	50	40	25	20
5	7	0.7	4	70	80	49	56
6	10	1	3	70	90	70	90
7a	10	1	2	20	40	20	40
7b	10	1	2	20	30	20	30
8	8	0.8	1	30	30	24	24
9	10	1	2	50	70	50	70
10	2	0.2	3	70	80	14	16
11	5	0.7	3	50	60	35	42
12	10	1	4	30	30	30	30

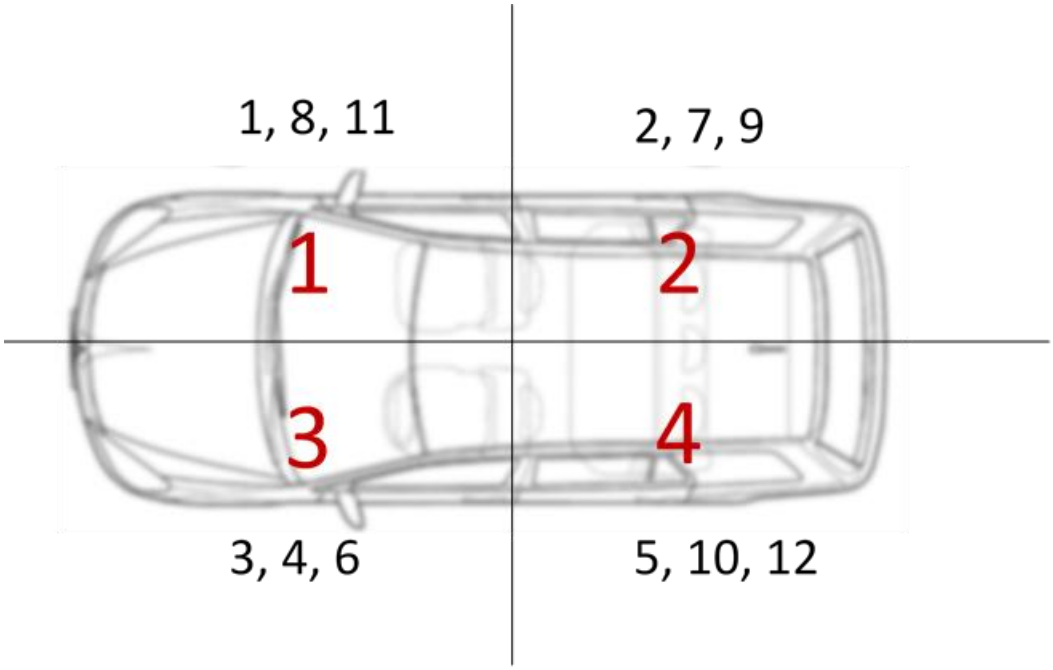

Figure 6: Mounting position for each task

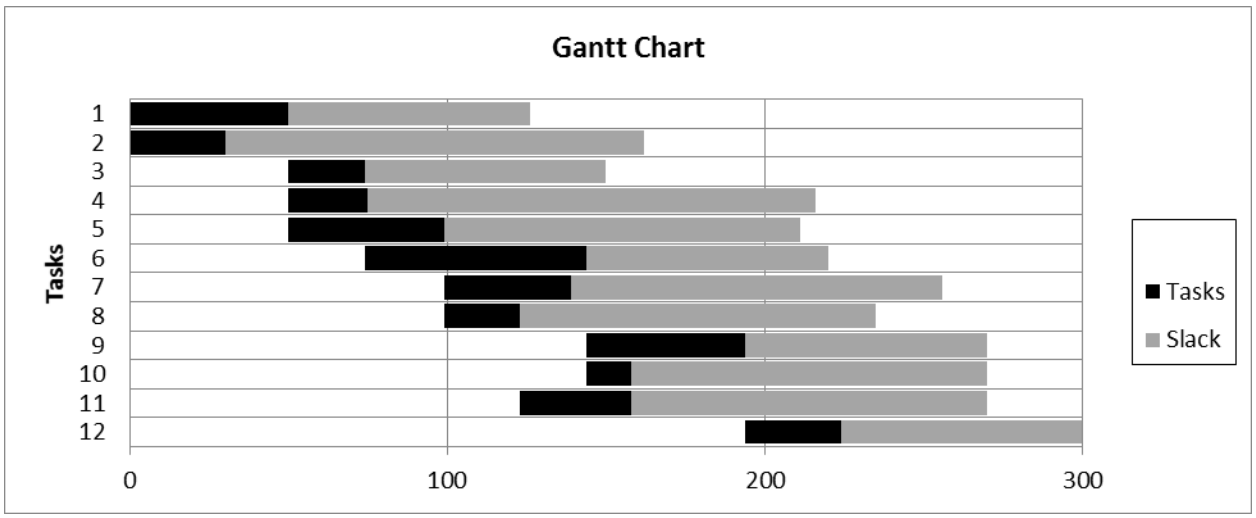

Figure 7: Gantt chart with tasks at early start times

Table 2: Assignment constraints

Assignment Constraint	**Tasks**
Adjacency set	(7a,7b)
Same worker	(9,12)
Same station	(4,5)
Not the same worker	$\{(h,l): q_h, q_l = (1,4),(2,3)\}$
Not the same station	(10,11)

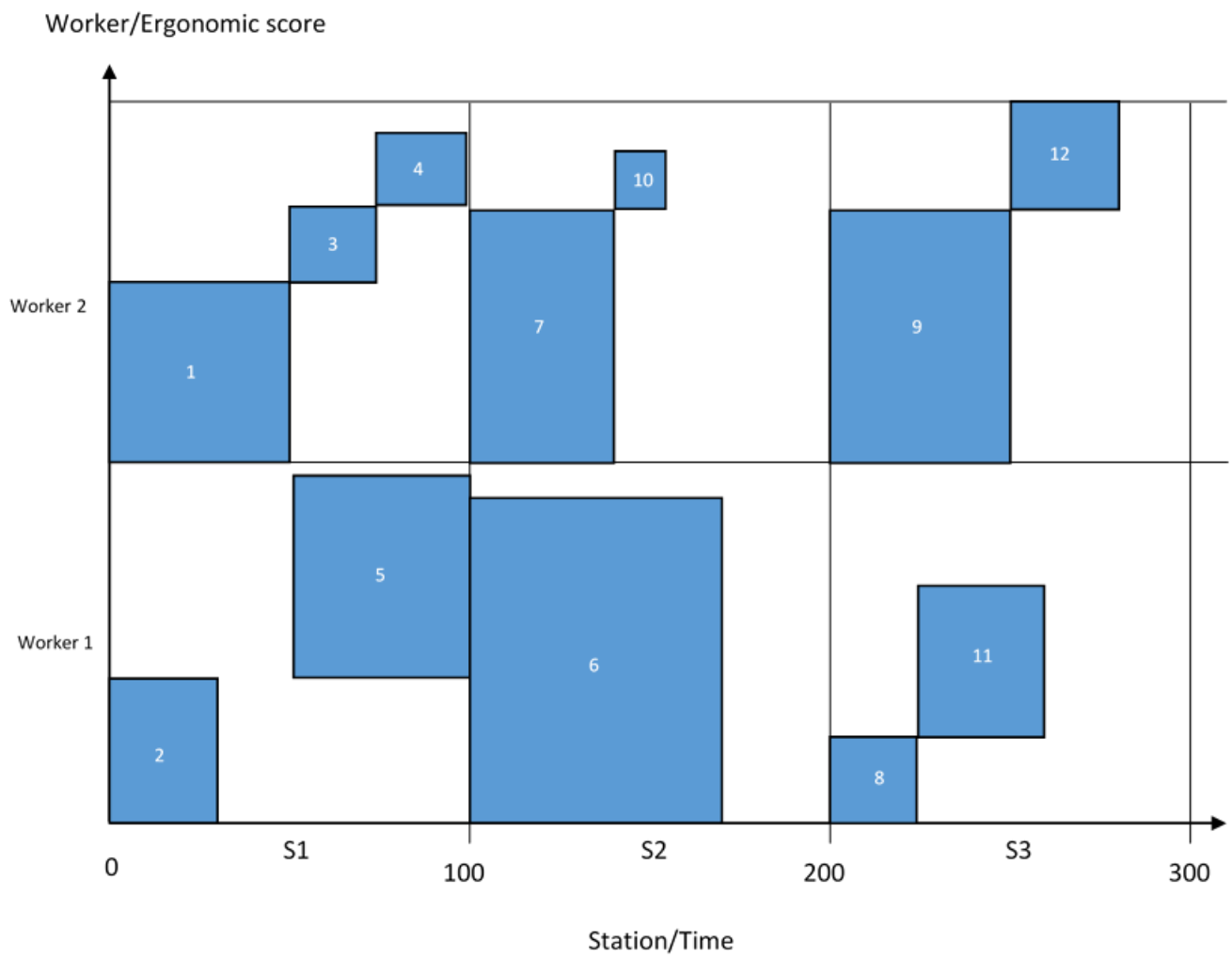

Figure 8: An optimal solution to the example

2.3 Related work

The basic mixed model assembly line balancing problem (MALBP) is based on the same assumptions that are used for the SALBP. The main difference is having multiple models each with its own precedence graph and task times. In the MALBP literature, two basic approaches are used to model and solve the problem: reduction to single model problem and horizontal balancing (Becker and Scholl, 2006).

The MALBP can be transformed to the SALBP by the use of a joint precedence graph. This graph is constructed by averaging the processing times that varies across models taking into account the probability of having each model in the expected model mix. The new joint precedence graph will have tasks with the expected processing times. This way the mixed model balancing problem is reduced to a single model case and traditional single model techniques can be used to solve it. With the increase of product variety in some production fields such as car manufacturing, reliable estimation of each model is becoming harder. In the paper by Pil and Holweg (2004) the number of model varieties that are available from different car manufacturers is shown; for some German car manufacturers, the theoretical variations can reach over 10^{24} different models. With this huge number of models, getting an estimation for the demand of each model is impossible. Boysen et al (2009) propose the use of the occurrence of the options instead of the model in determining the task times of the joint precedence graph. However, this requires the assignment of tasks to options and not just models. Van Zante-de Fokkert and De Kok (1997) review the two approaches of transforming the mixed model problem into a single

model: the combined precedence diagram approach and the adjusted task processing time approach.

Gokcen and Erel (1997) develop a binary goal programming (GP) model for the MALBP. The authors claim that they are the first to apply the multiple criteria decision making approach to the MALBP which gives the decision maker a more realistic approach and the ability to use different conflicting objectives in the model. Although the optimal solution may not be identified through this solution procedure, a satisfactory result that compromises the different conflicting objectives is provided. In addition, Gokcen and Erel (1998) propose a binary integer programming model for the MALBP along with some computational properties of the model. They claim that their model is superior to the other in the literature in terms of decision variables numbers and constraints. Furthermore, a shortest route formulation for the MALBP is presented by Erel and Gokcen (1999). The formulation is based on the shortest route formulation for the single model problem, thus the reduction of the mixed model to a single model by the combined precedence graph is required for this formulation.

Sawik (2002) compares two approaches to solve the combined balancing and scheduling problem for the flexible assembly line. In the monolithic approach, the balancing and assembly decisions are made simultaneously using a mixed integer program. For the hierarchical approach, the station workloads are balanced first and then the assembly process is scheduled by solving a permutation flow shop problem. Öztürk et al. (2013) also study the problem of balancing and scheduling in mixed model assembly lines

with parallel stations. They formulate the problem as a mixed integer program and then propose a decomposition scheme to be applied for large scale applications.

Some research introduces the multi-objective (MO) assembly line balancing problem such as the multi-objective approach proposed by Kara et al. (2011) to solve the MALBP for model mixes that have precedence conflicts. They develop a binary mathematical model with a single objective and then extend the model to incorporate three objectives into a single objective model. Mahdavi et al. (2009) propose a fuzzy multi-objective linear programming model for solving multi-objective MALBP. Their model makes use of a two-phase linear program: the first using a max-min approach and the second using the max-min solution as a lower bound to maximize the composite satisfaction degree.

2.3.1 Exact methods

Most of the literature on exact solution methods is focused on the SALBP as only a few propose exact methods for the MALBP. To use the single model solution methods on the mixed model problem, the mixed model need to be reduced to a single model by the use of combined precedence graph, after which modified single model techniques can be applied. Swell and Jacobson (2012) present an exact algorithm for the SALBP named the branch, bound and remember algorithm. The authors claim that this algorithm manages to find the optimal solution for all combined benchmark problems of Hoffmann, Talbot, and Scholl. The proposed algorithm is a hybrid method that combines branch and bound with dynamic programming. Scholl and Becker (2006) review state of the art exact and heuristic methods used to solve the SALBP. An important point to note is that although solving the

average SALBP guarantees that the line balance will not violate the cycle time on average, the optimal solution might still have significant inefficiencies when put to practice. Scholl (1999) point out that there are some necessary modifications that need to be done on any SALBP exact solution when applying it to a MALBP. This could be done by adding a secondary objective for smoothed stations loads. However, for branch and bound based algorithms, all optimal solutions have to be considered and the node fathoming procedure needs to be updated. In addition, the search should not be limited to maximum station loads as the secondary objective may prefer otherwise.

Ege et al. (2009) propose a branch and bound algorithm to solve the assembly line balancing problem with station paralleling. Boysen and Fliedner (2008) propose a two stage graph algorithm (Avalanche) that is able to solve GALBP with relevant constraints. The authors claim that this approach can be easily modified to include extensions such as parallel work stations, processing alternatives, zoning restrictions, stochastic processing times and U-shaped assembly lines.

Vilà and Pereira (2014) study the assembly line worker assignment and balancing problem. They provide an exact enumeration procedure that is based on the branch-bound and remember algorithm presented by Swell and Jacobson (2012). The authors develop several lower bounds, reductions, and dominance rules for the problem.

Wilhelm and Gadidov (2004) develop two models to address different tooling requirements in the multiple product assembly system design problem. The authors propose a branch and cut approach that employs a facet generation procedure to generate cuts in the search tree by exploiting some special structures of the problem. The authors

demonstrate the use of the approach through experiments and compare it to the available solution procedures.

One of the assumptions for the MALBP states that identical tasks have to be assigned to the same station for all models. The problem can be decomposed into several SALBP (one for each model) by relaxing this assumption. Bukchin and Rabinowitch (2006) relax the assumption of assigning identical tasks across different models to the same station, which allows a common task to be assigned to different stations for different models. The authors develop an integer program formulation and an exact solution method based on a backtracking branch and bound algorithm. In their model, some costs are associated with assigning an identical task to different stations. Hence, they needed to modify the objective function as the goal is to minimize the total cost associated with balancing and not just minimizing the number of stations. However, according to Becker and Scholl (2006), this relaxation is not desired in practice for several reasons such as the complex production control, facility requirements, loss of specialization effects and setup inefficiencies.

2.3.2 Constraint programming models

There has not been much research in using CP to solve the assembly line balancing problem in the past although it is one of the most successful tools in solving combinatorial problems. Bockmayr and Pisaruk (2001) propose a hybrid approach for solving the SALBP by combining CP and IP. The main contribution in their paper is developing a branch and cut solver for the SALBP and to show how it can make use of the CP solver in pruning the search tree. Pastor et al. (2007) compare the performance of impulse variables based

models, step variable based models, CP models, and IP models in solving the SALBP with different objectives. The authors conclude that the CP formulation performs better and is faster than IP but the best results were for the impulse variable based models. Recently Öztürk et al. (2013b) propose a MIP and a CP model to solve the flexible mixed model assembly line balancing and sequencing problems. They compare the performance of the MIP, the complete and various decomposition methods, and CP. According to their computational study, the CP outperforms all the other approaches over all sizes of the test instances. To the best of our knowledge there is no other published research that uses CP in solving the mixed model assembly line balancing problem with parallel station, zoning constrains and ergonomics.

2.3.3 Heuristics and meta-heuristics

The bulk of the research in the solution procedures for the MALBP is based on heuristics and meta-heuristics. The main reason for that is the complexity of the problem as the SALBP is just a bin packing problem when the precedence constraints are removed. The bin packing problem is a known NP-hard problem, thus the MALBP is NP-hard since it is just a SALBP with more than one model. More details on the complexity of the problem is given in Scholl (1999).

2.3.3.1 Heuristics:

Matanchai and Yano (2001) propose an approach to solve the MALBP by using an objective that helps achieving a better short term workload stability. Based on this objective, the authors develop a heuristic solution procedure that is based on filtered beam search in which feasible subsets of tasks are constructed at each station. For every station,

several feasible subsets with the best objectives are retained. Potential subsets at subsequent stations are then constructed by branching off the subset with the best objective. If the process is stopped because of infeasibilities, backtracking is used. After a feasible solution is found, an improvement procedure is used to reassign tasks between stations to improve the objective function.

Bukchin et al. (2002) design a three-stage heuristic to solve the mixed model assembly line design problem in a make to order environment. The heuristic minimizes the number of stations given a predetermined cycle time through stages: balancing using combined precedence graph, balancing for each model given constraints from the first stage, and neighborhood search. The authors relax the assignment constraint by allowing identical tasks across models to be assigned to different stations.

Hop (2006) solve the fuzzy MALBP in which processing time is fuzzy. The author propose a heuristic that aggregates fuzzy time, and used the combined precedence diagram to transfer the problem into a fuzzy SALBP. In addition, the author develop new approximated fuzzy arithmetic operations to calculate fuzzy numbers and then formulated the problem as a mix-integer program. The heuristic proposed is based on a flexible exchange sequence procedure to assign tasks into workstations.

Tonelli et al. (2013) propose a mixed integer program model that is solved by an iterative heuristic which aims at solving a number of aggregate planning problems in a mixed model production environment. The iterative heuristic is used to solve linear relaxed problems and reduced mixed integer program problems. The proposed optimization

approach combines the use of mixed integer program solver with a rolling horizon decomposition heuristic to solve practical problems.

McMullen and Frazier (1997) present a heuristic to solve a stochastic MALBP with task paralleling. The authors compared different task selection rules through simulation experiments. Sparling and Miltenburg (1998) propose an approximate solution algorithm for the U-shaped MALBP. The heuristic is based on a branch and bound algorithm applied to the combined precedence graph, followed by a smoothing algorithm to smooth the balance.

2.3.3.2 Meta-heuristics:

Tsujimura et al. (1995) are the first to use a genetic algorithm (GA) with a GALBP in which processing time is fuzzy. Falkenauer (1998) propose in his book a modified version of the GA named the grouping GA in which he shows the advantages of applying it to grouping problems including the assembly line balancing problem. Simaria and Vilarinho (2004) develope an iterative GA based procedure and applied it to the MALBP with parallel stations. The authors claim that their procedure accommodates other extensions such as zoning constraints and workload balancing. Recently, Sivasankaran and Shahabudeen (2013) use GA to solve the MALBP without converting the problem to a SALBP. That is, they used the original task times for each model in determining the line balance. In addition to the work presented, GA was used to solve the SALBP in many studies. More details on the use of GA to solve assembly line balancing problem is given in the survey by Tasan and Tunali (2008).

Vilarinho and Simaria (2002) develop a two-stage procedure using simulated annealing (SA) to solve the MALBP with zoning constraints and parallel workstations. The first stage searches for a sub-optimal solution to the primary objective which is minimizing the number of stations while the second stage deals with the secondary objective of smoothing workload across stations. Özcan and Toklu (2009) propose a SA algorithm to solve the mixed model two-sided line balancing problem by considering two objectives: minimizing the line length and minimizing the number of workers. Two performance measures are used: maximizing the weighted line efficiency and minimizing the weighted smoothness index. Manavizadeh et al. (2013) develope a three-stage SA algorithm to solve the mixed model U-line assembly line balancing problem in a Just in Time (JIT) production system. The stages are: solving the balancing problem to determine the number of stations, solving a worker assignment problem, and designing an alert system that is based on the Kanban system to balance the work in process inventory.

Vilarinho and Simaria (2006) revisit the same problem that they studied in (2002) but using an ant colony optimization (ACO) algorithm that searches for solutions in which the workload among workstations is smoothed. The authors claim that the results obtain using ACO were better than results obtained in (2002) using SA. Simaria and Vilarinho (2009) develop another ACO for the two sided MALBP in which two ants work together to build a feasible balancing solution with respect to zoning, capacity, side and synchronism constraints. McMullen and Tarasewich (2003) use a heuristic that is based on ACO to solve the problem of MALBP with constraints such as parallel workstations and stochastic task times. The authors claim that the solution method developed exploits the

properties of the assembly line balancing problem and produces good solutions in a reasonable time.

Bock (2008) uses clustered Tabu search as a base for his distributed search approach to solve the integrated problem of mixed model line balancing, personal, and process planning. The idea behind this approach is to use the distributed search technique on a network of computers in order to solve complex problems.

Furthermore, hybrid heuristics have been used to solve the mixed model assembly line balancing problem. Noorul Haq et al. (2006) propose a hybrid GA approach to solve the MALBP in which the modified ranked positional solution is used as an initial solution to reduce the search space. Akpınar and Bayhan (2011) develop a hybrid GA algorithm to solve the MALBP with parallel workstations and zoning constraints. To overcome the GA's shortage of exploring the search space effectively, the approach sequentially hybridized three well known heuristics with GA. Akpınar et al. (2013) solve the MALBP with parallel workstations, zoning constraints, and sequence dependent setup times between tasks using a ACO-GA hybrid algorithm. The authors aimed at combining the power of diversification in the ACO with the power of intensification in the GA.

Meta-heuristics are also used in several multi-objective assembly line balancing work. McMullen and Frazier (1998) use SA to solve a multi-objective MALBP with parallel workstations. In addition to the traditional performance objectives, the authors study the effect of a combination of several search objectives on performance measures such as cycle time and design cost. McMullen and Tarasewich (2006) develop a technique that is derived from ACO to solve the multi-objective GALBP. Several objectives are

addressed simultaneously such as crew size, system utilization, and system design cost. Recently, Chutima and Chimklai (2013) present a special PSO algorithm to solve the two-sided multi-objective MALBP. The primary objectives include mated stations and number of stations and the conflicting secondary objectives are workload relatedness and smoothness.

The summary of the extensions and methodology used in related literature is shown in Table 3.

Table 3: Summary of related literature

Researchers (Year)	**Mixed model**	**Eligible stations**	**Tooling**	**Task (Worker/Station) compatibility**	**Parallel stations**	**Worker-interference zoning**	**Ergonomics**	**Methodology**
Akpınar & Bayhan (2011)	✓			✓	✓			Hybrid GA
AkpıNar et al. (2013)	✓			✓	✓			ACO-GA
Becker & Scholl (2009)		✓		✓	✓	✓		B&B, Heuristics
Bock (2008)	✓	✓						TS
Bukchin et al. (2002)	✓							Heuristics
Bukchin et al. (2006)	✓							B&B, Heuristics
Chutima & Chimklai (2012)	✓							PSO
Ege et al. (2009)			✓		✓			B&B, Heuristics
Erel (1999)	✓							Shortest path model
Gokcen & Erel (1997)	✓	✓		✓				GP
Gokcen & Erel (1998)	✓	✓						Binary IP
Haq et al. (2006)	✓							Hybrid GA
Hop (2006)	✓	✓						Fuzzy binary LP
Kara (2011)	✓							MO GP
Mahdavi et al. (2009)	✓							MO fuzzy LP
Manavizadeh et al. (2013)	✓							SA
McMullen & Frazier (1997)	✓				✓			Heuristics

McMullen & Tarasewich (2003)	✓				✓			ACO
McMullen & Tarasewich (2006)								MO ACO
Otto & Scholl (2011)							✓	Heuristics
Özcan Toklu (2009)	✓							SA
Öztürk et al. (2013)	✓	✓			✓			Decomposition
Öztürk et al. (2013b)	✓	✓			✓			CP
Sawik (2002)	✓							IP
Simaria & Vilarinho (2004)	✓			✓	✓			GA
Simaria & Vilarinho (2009)	✓			✓				ACO
Sivasankara & Shahabudeen (2013)	✓							GA
Tsujimura et al. (1995)								GA
Vilà & Pereira (2014)								B&B
Vilarinho & Simaria (2002)	✓			✓	✓			SA
Vilarinho & Simaria (2006)	✓			✓	✓			ACO
Wilhelm & Gadidov (2004)	✓		✓					B&C, Heuristics

2.4 Motivation

The main motivation for this work is the fact that there is no applicable model that can be used to solve real life assembly line balancing problems. There is a gap between the research and the industry in the area of line balancing. Falkenauer (2005) discussed how the SALBP solution methods discussed in the literature has little or no application to the real world assembly line balancing problem. Furthermore, even the reported extensions in the general assembly line balancing problem literature are not applicable to real problems in the industry. The reason is that each extension usually tackles a generalization in one direction, while another research might consider other generalizations in another direction. What the industry really needs is a way to deal with all of these generalizations at the same time. The main characteristics needed to capture realism and make the solution applicable are summarized as follows:

- The majority of real life line balancing takes place at an already built assembly plants so the number of stations is fixed and re-balancing is what is needed.
- Each station has its own identity with the tooling and restrictions of the tasks that can be done on it.
- There are zoning constraints that should be considered in line balancing.
- Since the assembly plants are already built, elimination of stations is not feasible.
- Smoothing the workload across stations should be an important objective of the line balance.
- Multiple operators usually work in a station, so a schedule is needed to increase utilization and remove idle time.
- Station level ergonomic risks should be considered and might lead to assignment restrictions.

Also incorporating worker level ergonomic risks such as avoiding work area interference between workers should be incorporated.

- In a multi-product line, care should be taken in computing the average task times, horizontal balancing should be applied, and handling of drifting operations.

To the best of our knowledge, no published research has considered all of these characteristics simultaneously. Thus there is a need for a solution approach that takes most if not all of these realistic aspects into consideration.

2.5 The mathematical model

The integer program formulation for this problem is based on the formulation by Becker and Scholl (2009) for the single model variable parallel workstations assembly line balancing problem. In our model, it is assumed that tasks have a duration that is less than the cycle time so there is no need to divide tasks between stations. Also, mounting positions are not assigned to workers so several workers in the same station can be assigned tasks that share the same mounting position. Detailed scheduling is used to make sure that worker interference does not occur.

Each product assembled on the line starts with a work-piece that is launched down the line and have assembly tasks done on it until the work-piece exits the final station as a finished product. The work-piece enters the first station at time $t = 0$ and leaves the last station at time $t = Kc$. For each of the $i = 1, \dots K$ stations in the line there are $j = 1, \dots, W$ potential workers available, so worker (i, j) represents the potential worker j at station i. The binary variable $y_{i,j}$ is associated with each potential worker (i, j) and is equal to 1 if the potential worker is assigned and 0 otherwise. Figure 9 shows an example profile for a line with 3 stations and 4 workers, the greyed blocks represent the assignment of potential worker (i, j) when $y_{i,j} = 1$. In this example 3, 2, and 4 workers are assigned to stations 1, 2, and 3 respectively. The model's notations are shown in Table 4.

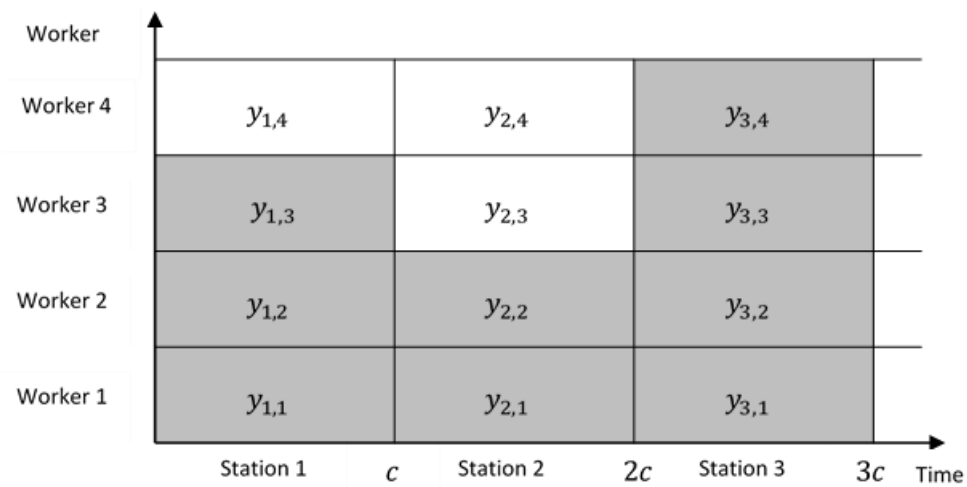

Figure 9: Example for a worker/station profile

Notations:

Table 4: Line balancing problem's notations

K	Maximum number of stations indexed $i = 1,2,3, \dots, K$
W	Maximum Number of potential workers per station indexed $j = 1,2,3, \dots, W$
N	Number of tasks indexed $h, l = 1,2,3, \dots, N$
c	Cycle time
A_i	Starting time of station i $\quad A_i = (i-1) * c$
t_h^s	Early start time for task h
t_h^f	Late finish for task h
q_h	Mounting position of task h
p_h	Proportion of models that will require task h in the production run
d_h	Duration of task h
$\bar{d}_h$	Weighted duration of task h
g_h	Ergonomic score for task h
$\bar{g}_h$	Weighted ergonomic score for task h
G	Ergonomic risk score limit
F_h	The set of feasible stations that task h is assignable to
U_h	The set of stations with the required resources to perform task h
O_h	The set of immediate predecessors of task h
R^{ss}	The set of tasks that must be done on the same station
R^{ns}	The set of tasks that cannot be done on the same station
R^{sw}	The set of tasks that must to be done by the same worker
R^{nw}	The set of tasks that cannot be done by the same worker

Decision Variables:

$$x_{ijh} = \begin{cases} 1 & if\ task\ h\ is\ assigned\ to\ station\ i\ and\ worker\ j \\ 0 & otherwise \end{cases}$$

$$y_{ij} = \begin{cases} 1 & if\ potential\ worker\ j\ at\ station\ i\ is\ assigned \\ 0 & otherwise \end{cases}$$

$$v_{hl} = \begin{cases} 1 & if\ task\ h\ is\ executed\ before\ task\ l \\ 0 & otherwise \end{cases}$$

$$s_h = starting\ time\ of\ task\ h$$

Objective:

$$minimize\ Z = \sum_{i} \sum_{j} y_{ij}$$

The objective is to minimize the total number of workers assigned.

Subject to the following constraints:

- Task assignment to worker: each task has to be assigned to only one worker.

$$\sum_{i \in FS_h} \sum_{j} x_{ijh} = 1 \qquad \forall\ h \qquad \text{(C2.1)}$$

- Cycle time: the total weighted duration assigned to a worker should not exceed the cycle time.

$$\sum_{h} x_{ijh} \bar{d}_h \leq c\, y_{ij} \qquad \forall (i \in F_h\ , j) \qquad \text{(C2.2)}$$

- Station time: each task assigned to a worker should be scheduled between the worker's station start and finish times.

$$s_h \geq \sum_{i \in F_h} \sum_{j} S_i x_{ijh} \qquad \forall h \qquad \text{(C2.3)}$$

$$s_h + \bar{d}_h \leq \sum_{i \in FS_h} \sum_{j} (S_i + c) x_{ijh} \qquad \forall h \qquad \text{(C2.4)}$$

- Precedence relations: a task can only start when all of its predecessors are finished.

$$s_h + \bar{d}_h \leq s_l \qquad \forall\ h, l \in O_l \qquad \text{(C2.5)}$$

- Tasks overlap: all tasks assigned to a worker should not overlap.

$$v_{hl} + v_{lh} \geq x_{ijh} + x_{ijl} - 1 \qquad \forall j\ , h \neq l\ , and\ \ i \in F_h \cap F_l \qquad \text{(C2.6)}$$

$$s_h + t_h \leq s_l + (1 - v_{hl})(t_h^f - t_h^s) \quad \forall h \neq l \qquad \text{(C2.7)}$$

- Worker interference: tasks that share the same mounting position should not overlap.

$$v_{hl} + v_{lh} \geq \sum_j (x_{ijh} + x_{ijl}) - 1 \quad \forall i \in F_h \cap F_l \,, h \neq l \,, and\ q_h = q_l \qquad \text{(C2.8)}$$

- Ergonomic risk: the total weighted ergonomic risk for each worker should be within limits.

$$\sum_h x_{ijh}\, \bar{g}_h \leq G \qquad \forall (i \in F_h \,, j) \qquad \text{(C2.9)}$$

- Assignment restrictions: pairs of tasks with same (station/worker) or not the same (station/worker) restrictions.

$$\sum_j x_{ijh} = \sum_j x_{ijl} \qquad \forall i \in F_h \cap F_l \ and\ (h,l) \in R^{ss} \qquad \text{(C2.10)}$$

$$x_{ijh} = x_{ijl} \qquad \forall\ (i \in F_h \cap F_l \,, j)\ and\ (h,l) \in R^{sw} \qquad \text{(C2.11)}$$

$$\sum_j (x_{ijh} + x_{ijl}) \leq 1 \qquad \forall i \in F_h \cap F_l \ and\ (h,l) \in R^{ns} \qquad \text{(C2.12)}$$

$$x_{ijh} + x_{ijl} \leq 1 \qquad \forall\ (i \in F_h \cap F_l \,, j)\ and\ (h,l) \in R^{nw} \qquad \text{(C2.13)}$$

- Resource requirement: tasks that require a resource should only be assigned to eligible stations.

$$x_{ijh} = 0 \qquad \forall i \notin U_h \,, j, and\ h \qquad \text{(C2.14)}$$

- Breaking symmetry.

$$y_{ij} \geq y_{i(j+1)} \qquad \forall\ (i, j \leq W - 1) \qquad \text{(C2.15)}$$

- Variables domain.

$$x_{ijh}, y_{ij}, v_{hl} \in \{0,1\}, \quad s_h \in \mathbb{Z}^+ \qquad \forall i, j, h, and\ l \qquad \text{(C2.16)}$$

The original model formulation proposed by Becker and Scholl (2009) did not provide a good lower bound for the integer program. The introduction of the cycle time constraint (C2.2) provides a good lower bound for the integer program that is equivalent to the capacity bound (LB1) presented in their paper. The lower bound achieved by this constraint gives the lowest possible number of workers needed to perform all tasks which is equivalent to rounding up the sum of all weighted task durations divided by the cycle time.

$$LB = \left\lceil \sum_h \bar{d}_h / c \right\rceil \qquad (2.1)$$

In addition, the assumption that all tasks that share the same mounting position are to be assigned to the same worker is relaxed in this work. Any task can be assigned to any worker regardless of its mounting position. Reducing the unproductive waking from a position to another is achieved by prohibiting the same worker from being assigned tasks with non-contiguous mounting positions. To avoid worker interference, constraint (C2.8) is added to make sure that no two tasks that share the same mounting position overlap in time in any station. Constraints (C2.10) and (C2.13) are for the same station and same worker assignment restrictions constraints. The

original model uses the worker incompatible and station incompatible constrains only. Constraint (C2.14) helps in reducing the feasible space by fixing some decision variable by the use of the eligible station set for each task. Since all workers are identical, constraint (C2.15) reduces the problem's feasible region by removing the symmetry from the problem in an effort to reduce the computation time. For example, the constraint enforces the assignment of worker 1 before assigning worker 2.

2.6 The constraint programming model

Since it is anticipated that the mathematical programming model will face difficulties in finding a solution for real life sized problem instances, we propose a constraint programming (CP) model. CP is proven to be a useful technique for solving scheduling problems efficiently. Several tools exist for developing CP models; ILOG CPLEX optimization studio is one of the most used for scheduling problems because of the ILOG Scheduler extension that was developed specifically with several constraint propagation and search algorithms that exploit the structure of scheduling problems. ILOG Scheduler is used in this work to model the mixed model line balancing problem. Before presenting the formulation we first introduce some basic variable types used in the ILOG Scheduler environment. After that some specialized constraints used in the model are introduced. These specialized constraints state some relation between different variables in the model. Although these relations could be modeled using traditional logical constraints; the use of the specialized constraint increases the efficiency of the solver by exploiting the structure of the problem through the use of specialized filtering algorithm.

2.6.1 CP variables and special constraints

2.6.1.1 Interval variables

Interval variables represent an interval of time in which a task or an activity is carried out whose position in time is unknown and need to be scheduled. This decision variable is characterized by a start value, an end value and a size. Interval variables can be optional; that is not all variables would be considered in the solution. This is usually helpful in modeling optional tasks that doesn't need to be executed, tasks that can be executed on alternative resources, and tasks with alternative modes of operation. If an interval variable is optional it can be present or absent. An absent variable is not considered in any constraint or expression it is involved in.

2.6.1.2 Interval sequence variables

The interval sequence variable is defined over a set of interval variables and its value represents the ordering of the tasks in that set. Absent interval variables are not considered in this ordering. All the permutations of the intervals in the set are possible values for the interval sequence decision variable. It should be noted that this variable does not enforce any constraint on the relative position between interval variables but there are several constraints that can be used on this decision variable.

2.6.1.3 Specialized constraints

- Alternative constraint:

The $alternative$ constraint is used to create a constraint between an interval variable a and a set of optional interval variables $\{b_1, \dots, b_n\}$ such that if interval a is present then exactly one of the intervals $\{b_1, \dots, b_n\}$ is present. It is indicated as $alternative(a, \{b_1, \dots, b_n\})$

- The no overlap constraint:

The $noOverlap$ constraint facilitates modeling disjunctive resources and operates on sequence variables to define a chain of non-overlapping intervals. The intervals in the chain are scheduled such that any interval in the chain ends before the start of the next interval in the chain.

- Precedence relation constraint:

There are several constraints that can be used to restrict the relative position of the interval variables. One of these special constraints is the $endBeforeStart$ constraint that enforces the precedence relation between two interval variables such that the predecessor interval ends before the start of the successor interval.

- Logical constrains:

The presence status of interval variables can be controlled by using the logical constraint $presenceOf(a)$. For example, interval a can be forced to be present if interval b is present by using the following logical constrain:

$$presenceOf(a) \geq presenceOf(b)$$

2.6.2 The constraint programming model formulation

Decision variables:

$Task_h$: Interval variable associated with task h with a size equal to $\bar{d}_h$

$TaskAssign_{h,i,j}$: Interval variable associated with task h that is performed by worker j on station i, limited to lie within $\left[t_h^s, t_h^f\right] \cap [A_i, A_i + c]$

$Worker_{i,j}$: Sequence variable associated with worker j at station i

Objective:

Minimize $\sum_i \sum_j \max_h \; presenceOf(TaskAssign_{h,i,j})$

Subject to:

- Task assignment to worker: defining the alternative interval variable such that each task interval variable has an alternative variable for each worker at each station. Only one of the alternatives will be present; in other words each task is assigned to only one worker.

$$alternative(Task_h, TaskAssign_{\forall i, \forall j}) \qquad \forall h \qquad \text{(CP2.1)}$$

- Precedence relations: a task can start when all of its predecessors are finished.

$$endBeforeStart(Task_h, Task_l) \qquad \forall\, h, l \in O_l \qquad \text{(CP2.2)}$$

- Tasks overlap: tasks that are assigned to one worker should not overlap.

$$noOverlap(Worker_{i,j}) \qquad \forall i, j \qquad \text{(CP2.3)}$$

- Worker interference: tasks that share the same mounting position should not overlap.

$$noOverlap(Task_h) \qquad \forall h: q_h = 1,2,3, \dots, Q \qquad \text{(CP2.4)}$$

- Ergonomic risk: the total weighted ergonomic risk for each worker should be within limits.

$$\sum_h presenceOf(TaskAssign_{h,i,j})\, \bar{g} \leq G \qquad \forall\, i, j \qquad \text{(CP2.5)}$$

- Assignment restrictions: pairs of tasks with same (station/worker) or not the same (station/worker) restrictions.

$$\sum_j presenceOf(TaskAssign_{h,i,j}) = \\ = \sum_j presenceOf(TaskAssign_{l,i,j}) \qquad \forall\, j\ and\ (h, l)\ \in R^{ss} \qquad \text{(CP2.6)}$$

$$presenceOf(TaskAssign_{h,i,j}) = \\ = presenceOf(TaskAssign_{l,i,j}) \qquad \forall\, i, j, and\ (h, l)\ \in R^{sw} \qquad \text{(CP2.7)}$$

$$\sum_{j}(presenceOf(TaskAssign_{h,i,j}) + \sum_{j} presenceOf(TaskAssign_{l,i,j})) \leq 1 \quad \forall j \; and \; (h,l) \in R^{ns} \qquad \text{(CP2.8)}$$

$$presenceOf(TaskAssign_{h,i,j}) + presenceOf(TaskAssign_{l,i,j}) \leq 1 \quad \forall \; i,j, and \; (h,l) \in R^{nw} \qquad \text{(CP2.9)}$$

- Resource requirement: tasks that require a resource should only be assigned to eligible stations.

$$presenceOf(TaskAssign_{h,i,j}) == 0 \qquad \forall i \notin U_h \; , j, and \; h \qquad \text{(CP2.10)}$$

- Breaking symmetry:

$$\max_{h} \; presenceOf(TaskAssign_{h,i,j+1}) \leq \max_{h} \; presenceOf(TaskAssign_{h,i,j}) \quad \forall \; (i,j \leq W-1) \qquad \text{(CP2.11)}$$

2.7 Computational experiments

We perform computation experiments to compare the performance of the CP and IP in solving industrial sized assembly line balancing problems. Data from the automotive industry is used to define a test bed. The data belongs to 3 different bands of actual auto assembly lines, Details on the size of each band before and after the preprocessing phase is provided in Table 5. Four instances generated from each band's data with different number of tasks (50, 100, 150, and all tasks) and different number of stations. Details on test instances from all bands are provided on Table 6. The same data is used in the dissertation by Pearce (2015).

The IP model is coded in AMPL and solved using Gurobi 6 solver. The CP is coded under the ILOG CPLEX optimization studio 12.6 and solved using the CPLEX CP Optimizer engine. The computer has an Intel i7-3770 CPU with 3.4 GHz clock speed and 32 GB of memory. The solution time is limited to one hour; if the optimal solution is not provably found, the best solution is recorded along with the time at which the best solution was reached. For CP, the lower bound is similar to the IP one which is equivalent to LB in (5.1). The search in CP is terminated when the solution is equal to the lower bound. Since CP search is based on random number generation, 10 runs were done for each instance with different seeds and the average solution time is recorded. The default inference level is set to "extended" in the CP along with the precedence, interval sequence, and no overlap inference levels.

Table 5: Number of tasks in each band

Band	**Original number of tasks**	**Number of tasks after pre-processing**
Band 1	395	209
Band 26	317	174
Band 47	408	178

Table 6: Test instances

Instance	Number of tasks	Number of stations
B1.1	50	5
B1.2	100	10
B1.3	150	10
B1.4	209	13
B26.1	50	5
B26.2	100	6
B26.3	150	10
B26.4	174	10
B47.1	50	5
B47.2	100	7
B47.3	150	11
B47.4	178	12

Table 7 summarizes the results of both the IP and CP subject to a one hour time limit. Since 10 runs were executed for each instance in CP, the number of runs the given solution (number of workers) was found is also shown. As expected the IP faced difficulties as the size of the instance increases. The optimal solution is found only in 3 small sized instances while no feasible solutions were found on the largest instance of each band within the one hour time limit. On the other hand, the CP managed to find optimal solutions in 10 out of the 12 instances with only a small gap in the solution of the other two instances. In CP instance B1.3 two different solutions were found for different seed runs, of which one is optimal. It is clear that Band 1 instances were difficult for both the CP and IP solvers. Table 8 presents the processing time needed by the IP solver to reach the optimal solution for different instances under no time limits. Results show that CP clearly outperforms the IP in terms of processing time making it more applicable in the industry. Table 8 shows the time it takes the IP to reach the optimal solution when there is no time limit, in which some instances utilized all memory before finding an optimal solution.

Table 7: Solution time under time limit

Instance	LB	IP (3600 sec time limit)				CP (3600 sec time limit)			
		Solution (# of workers)	**Solution time (sec.)**	**Solution (# of workers)**	**# found**	**Min solution time (sec.)**	**Max solution time (sec.)**	**Avg solution time (sec.)**	**SD Of solution time (sec.)**
B1.1	7	**7**	34.36	**7**	10	34.92	35.03	34.96	0.03
B1.2	12	15	3208	13	10	317.13	345.21	327.24	7.61
B1.3	16	-	-	**16** 17	6 4	509.245 442.61	3970.34 505.984	2263.94 468.03	1439.80 30.48
B1.4	22	-	-	23	10	1122.64	3347.13	1684.64	649.39
B26.1	5	**5**	13.47	**5**	10	34.11	34.68	34.30	0.23
B26.2	9	11	2030	**9**	10	100.47	110.16	104.98	3.19
B26.3	12	20	2552	**12**	10	446.15	807.44	534.73	116.33
B26.4	13	-	-	**13**	10	291.23	485.13	354.43	56.83
B47.1	6	**6**	17.83	**6**	10	28.41	28.47	28.45	0.02
B47.2	11	12	720	**11**	10	135.54	137.15	136.38	0.50
B47.3	16	-	-	**16**	10	492.02	717.79	555.20	69.22
B47.4	18	-	-	**18**	10	805.72	972.31	842.93	51.25

Table 8: IP solution time under no time limit

Instance	IP time to optimal solution (days, HH:MM:SS)
B1.1	00:00:34
B1.2	1 days, 21:43:19
B1.3	1 days, 19:01:49
B1.4	-
B26.1	00:00:13
B26.2	10:28:52
B26.3	12:37:00
B26.4	-
B47.1	00:00:17
B47.2	05:05:43
B47.3	3 days, 01:19:43
B47.4	-

2.8 Conclusion

In this chapter, the model by Becker and Scholl (2009) is extended to the mixed model environment with additional constraints from Falkenauer (2005) to improve the model's applicability to real mixed model assembly lines. First, we explain each constraint incorporated into the model and illustrate the problem at hand with an example. Then, we propose an IP formulation to solve the mixed model assembly line balance with parallel stations, zoning constraints, and ergonomics. Since the problem is NP-hard, we develop a scheduling CP model to solve industrial sized assembly line problems.

To test the model's applicability to real assembly lines, the data of three bands from an automotive assembly line were used. The results show the limitation of the IP, in which only instances with 50 tasks were solved to optimality within the allowed 1 hour of processing time. On the other hand, the CP model's results were optimal for 10 out of the 12 instances under the same time limit. For example, the CP managed to balance 178 tasks over 12 stations for band 47. The computational results show the potential of CP scheduler as a tool to solve real life sized assembly line balancing problems.

This work can be extended in future research by incorporating the idea of horizontal balancing and model sequencing to the mixed model line balancing problem. If a worker is expected to have a notably different amount of time to work on different models, his station might not be able to manage the work overload caused by having several models with time consuming tasks since only the average task duration satisfies the cycle time constraint. There are two ways to address this problem: horizontal balancing and model sequencing. Horizontal balancing makes the workload of each station (with respect to models/options) as uniform as possible over time regardless of the product sequence. This can be seen as a robust mixed model line balance that

minimizes work overload regardless of the sequence used. On the other hand, this problem can also be avoided by solving a sequencing problem that prohibits having multiple demanding models in a consecutive sequence and thus avoids any work overload. This can be achieved by developing an integrated model that solves real life sized line balancing / sequencing problems simultaneously.

CHAPTER THREE

3 A CASE STUDY

3.1 Introduction

This chapter explores the models developed for the assembly line balancing problem presented in chapter two through an increased sensitivity analysis using the case study data supplied by an automaker. In generating the data for the case study, the research team learned that some data is relatively easier or harder to obtain and maintain, and some constraints may not be discovered until a potential solution is presented that violates the constraints. Requiring an OEM to maintain large databases that are needed for only application to line balancing may increase the potential cost. In practice, satisfying the late-discovered constraints can sometimes be accomplished by a small permutation of the current solution, or the decision maker may determine that this or another constraint can be violated. The nature of the mathematical programming techniques used in this dissertation is to enforce all constraints provided, even if the decision maker considers some of them to be less-critical. The motivation is to build intuition about the relative cost of modeling, data collection and maintenance required to enforce some of the realistic constraints in the mixed model line balancing problem.

Some of the line balancing constraints define the problem at a fundamental level and cannot be ignored or relaxed. These constraints are items such as assigning each task to one worker, cycle time, task precedence relations, and tasks overlap constraints. On the other hand, some other constraints might be amenable to relaxation or removal such as worker interference (mounting), ergonomic risk, the worker/station compatibility assignment constraints, and the eligibility constraints. It maybe that some elements that are described as constraints are in fact preferences, and can be accommodated partially. If such constraints, when included in the model, are

computationally difficult, it may be acceptable to remove them and manually accommodate them later. Moreover, some constraints may not be acknowledged until a balance is proposed, at which time a manual readjustment is made, instead of reformulating and resolving. By studying the effects of relaxing or removing some of these constraints on the model behavior, we gain more insight on the complexity each constraint contributes to the model so that decision makers can make better use of it. In the following section a classification of the different line balancing constraints is given.

3.2 The line balancing constraints

We classify the line balancing constraints as hard or soft constraints. A constraint is labeled "hard" if removing it will either violate any of the SALBP assumptions or fundamentally affect the way the problem is modeled. On the contrary, a soft constraint will not violate any of the SALBP assumptions or fundamentally affect the model if removed.

3.2.1 Hard constraints

1. Task to worker assignment:

This constraint ensures that every task is assigned to one worker only.

2. Cycle time:

This constraint ensures that the total expected task durations assigned to a worker does not exceed the cycle time.

3. Station time:

This constraint defines each station's time window so tasks scheduled on that station are limited to start at or after the start of that station and finish before or at the finish time of that station.

4. Precedence relation:

The precedence relation between assembly tasks is a hard constraint that cannot be relaxed. Without the precedence constraint the problem can be transformed into a bin-packing problem in which the goal is to assign (pack) as many tasks to every station (bin) such that the number of stations is minimized.

5. Tasks overlap:

This constraint prohibits two tasks that are assigned to a worker from overlapping in time. That is, the worker should finish a task before starting another one.

3.2.2 Soft constraints

1. Same mounting position worker interference (mounting):

This constraint prohibits schedules in which two workers work on the same mounting position at the same time. That is, no two tasks that share the same mounting position overlap in time. This constraint can be relaxed assuming that the interference can either be avoided by manually changing the station's schedule or by communication between workers in the same station. These constraints are either enforced or not.

2. Task assignment restrictions constraints:

This includes same worker/station and not same worker/station constraints in addition to the task/station eligibility constraints. These constraints are either enforced or not.

3. Ergonomic risk:

This constraint limits the maximum possible weighted ergonomic score for the tasks assigned to any worker. This constraint can take different levels of maximum ergonomic score, or be completely eliminated.

3.3 Experimental configuration

The experiments use data provided by an OEM for 3 bands in their assembly line. The summary of the data sets used with the number of tasks along with the number of different constraints is given in Table 9. The experiments are conducted using both the IP and CP models developed previously.

It should be noted that because of the IP's limitation, the maximum number of tasks is limited to 100 in order to reach a solution in reasonable time. The solution time of the IP is limited to a day and 3600s for the CP; if no optimal solution is found the best solution is recorded along with the time it took the IP or CP to reach that solution. The CP experiments are run 10 times for each instance and the average is recorded along with the number of times the solution was found.

Table 9: Test data sets

Band	**# tasks**	**Number of stations/ workers**	**Task assignment constraints**					**Mounting zone constraints**
			Same station	**Same worker**	**Not same station**	**Not same worker**	**Eligibility**	
Band 1	50	5/25	0	9	0	102	8	640
Band 26	75	5/25	2	2	0	88	34	1896
Band 47	100	7/35	0	0	2	332	14	650

There are two main experiments to assess the models' sensitivity. The first experiment varies the maximum weighted ergonomic score allowed for each worker and recording both the solution and computation time. The second experiment is conducted by disabling a subset of constraints and recording the solution along with the computation time. Because the CP performance is based on the random number used, each CP instance is run for 10 times with random number seeds from 0 to 9. Both of these experiments are done on an i7-4790 CPU with 32 GB on Windows 10 platform. For the IP, AMPL Version 20161220 is used along with Gurobi 7.0.2 solver. For the CP, IBM ILOG CPLEX Optimization Studio Version 12.7.0 is used.

3.4 Results

3.4.1 Experiment 1: Sensitivity to maximum ergonomic score

In this experiment, the maximum ergonomic score allowed is varied depending on the tasks' ergonomic scores for each band. Table 10, Table 11, and Table 12 show results for bands 1, 26, and 47 respectively. In these tables, the maximum ergonomic score used is shown along with the IP's lower bound (LB) on the solution (number of workers), the IP solution (optimal is bolded), the computation time to reach this IP solution, the CP solution(s) (optimal is bolded), the number of time the CP solution is found, and the average CP computation time.

The range of possible maximum ergonomic scores for the different bands increases as the problem size increases. This range might be limited by the possible number of workers in each station. It should be noted that for each band there is a setting for the maximum ergonomic score in which the IP fails to reach the optimal solution even though values below and above this setting allows optimal solutions to be found. For these hard instances, the CP finds the optimal solution in some of the runs with increased computational time. This indicates that the problem becomes harder for some imposed maximum ergonomic levels as the solution pool becomes smaller. Results also show that relaxing the maximum ergonomic score does not always help in reducing the computation time.

Table 10: Sensitivity to maximum ergonomic score in Band 1

Exp (Band1)	**Max Ergo.**	**IP**			**CP**		
		LB	**Solution**	**Time (HH:MM:SS)**	**Solution**	**# found**	**Avg time (HH:MM:SS)**
E1.1	155	9	**9***	00:03:59	**9**	10	00:00:49
E1.2	160	8	**8**	00:00:45	**8**	10	00:00:50
E1.3	165	8	**8**	00:01:01	**8**	10	00:00:53
E1.4	170	8	**8**	00:00:33	**8**	10	00:00:47
E1.5	175	7	8	00:00:11	**7** 8	7 3	00:38:56 00:00:38
E1.6	180	7	**7**	00:00:33	**7**	10	00:01:33
E1.7	No limit	7	**7**	00:00:07	**7**	10	00:00:36

* Optimal solution is bolded

Table 11: Sensitivity to maximum ergonomic score in Band 26

Exp (Band26)	Max Ergo.	IP			CP		
		LB	Solution	Time (HH:MM:SS)	Solution	# found	Avg time (HH:MM:SS)
E26.1	150	12	**12**	00:05:43	**12**	10	00:00:57
E26.2	170	10	11	00:02:38	**10** 11	3 7	00:23:54 00:00:54
E26.3	180	10	**10**	00:09:20	**10**	10	00:01:08
E26.4	200	9	**9**	00:10:07	**9**	10	00:00:57
E26.5	250	7	8	00:00:49	**7**	10	00:09:00
E26.6	350	7	**7**	00:05:13	**7**	10	00:01:22
E26.7	No limit	7	**7**	00:08:26	**7**	10	00:01:08

Table 12: Sensitivity to maximum ergonomic score in Band 47

Exp (Band 47)	Max Ergo.	IP			CP		
		LB	Solution	Time (HH:MM:SS)	Solution	# found	Avg time (HH:MM:SS)
E47.1	90	32	**32**	06:47:13	**32**	10	00:02:17
E47.2	100	29	**29**	08:03:06	**29**	10	00:02:17
E47.3	110	26	27	05:32:34	**26** 27	5 5	00:02:18 00:02:26
E47.4	120	24	**24**	05:50:04	**24**	10	00:02:11
E47.5	130	22	**22**	08:02:31	**22**	10	00:02:14
E47.6	140	21	**21**	05:21:45	**21**	10	00:02:14
E47.7	150	19	**19**	10:03:30	**19**	10	00:02:25
E47.8	160	18	**18**	03:19:44	**18**	10	00:02:10
E47.9	170	17	**17**	07:05:31	**17**	10	00:02:12
E47.10	180	16	**16**	10:39:56	**16**	10	00:02:13
E47.11	200	15	**15**	05:12:00	**15**	10	00:02:13
E47.12	300	11	**11**	09:02:07	**11**	10	00:02:15
E47.13	500	11	**11**	11:32:06	**11**	10	00:02:15
E47.14	No limit	11	**11**	07:51:35	**11**	10	00:02:18

3.4.2 Experiment 2: Sensitivity to constraints

In this experiment, the effect of disabling one (or more) soft constraints on the solution quality and computation time is studied. For each band, a set of 8 different instances is generated. The first instance is created by disabling all three types of constraints, namely the ergonomics, mounting and assignment constraints. In the next three instances, only one type of constraints is disabled. The following three instances study the interaction between the different constraints by disabling two of them at a time. The last instance is the problem with no constraints disabled for comparison.

Table 13, Table 14, and Table 15 summarize the constraints sensitivity results for Bands 50, 26, and 47 respectively.

Table 13: Band 1 sensitivity to constraints

Exp	Constraints			IP			CP	
	Ergonomics (Max=165)	Mounting	Assignment	LB	Solution	Time (HH:MM:SS)	Solution	Avg time (HH:MM:SS)
S1.1	×	×	×	7	**7**	00:00:08	**7**	00:00:36
S1.2	✓	×	×	8	**8**	00:03:08	**8**	00:00:37
S1.3	×	✓	×	7	**7**	00:01:05	**7**	00:00:34
S1.4	×	×	✓	7	**7**	00:00:10	**7**	00:00:37
S1.5	✓	✓	×	8	**8**	07:37:56	**8**	00:00:40
S1.6	✓	×	✓	8	**8**	00:00:51	**8**	00:00:52
S1.7	×	✓	✓	7	**7**	00:00:07	**7**	00:00:36
S1.8	✓	✓	✓	8	**8**	00:01:01	**8**	00:00:53

Table 14: Band 26 sensitivity to constraints

Exp	Constraints			IP			CP	
	Ergonomics (Max=200)	Mounting	Assignment	LB	Solution	Time (HH:MM:SS)	Solution	Avg time (HH:MM:SS)
S26.1	×	×	×	7	**7**	00:04:59	**7**	00:00:55
S26.2	✓	×	×	9	**9**	02:28:42	**9**	00:00:56
S26.3	×	✓	×	7	8	00:03:58	**7**	00:01:21
S26.4	×	×	✓	7	**7**	00:00:51	**7**	00:00:55
S26.5	✓	✓	×	7	10	00:04:08	**7**	00:01:02
S26.6	✓	×	✓	9	**9**	01:07:20	**9**	00:00:52
S26.7	×	✓	✓	7	**7**	00:08:26	**7**	00:01:09
S26.8	✓	✓	✓	9	**9**	00:10:07	**9**	00:00:57

Table 15: Band 47 sensitivity to constraints

Exp	Constraints			IP			CP	
	Ergonomics (Max=200)	Mounting	Assignment	LB	Solution	Time (HH:MM:SS)	Solution	Avg time (HH:MM:SS)
S47.1	×	×	×	11	**11**	12:34:17	**11**	00:02:07
S47.2	✓	×	×	15	**15**	11:53:30	**15**	00:02:05
S47.3	×	✓	×	11	**11**	26:06:38	**11**	00:02:12
S47.4	×	×	✓	11	**11**	03:32:52	**11**	00:02:04
S47.5	✓	✓	×	15	16	10:56:52	**15**	00:02:04
S47.6	✓	×	✓	15	**15**	01:17:59	**15**	00:02:02
S47.7	×	✓	✓	11	**11**	07:36:31	**11**	00:02:08
S47.8	✓	✓	✓	15	**15**	05:12:00	**15**	00:02:13

There is an obvious trend in computation time increase for both the IP and CP as the number of tasks increase from 50 to 75, then to 100 across the different bands. Removing the assignment constraints in Band 1 increased the IP time to reach an optimal solution drastically. In addition, the IP was unable to find an optimal solution within a day after removing the assignment constraints in bands 26 and 47. On the contrary, solving the problem with only the mounting constraints increased the computation time in band 47; no optimal solution was reached in band 26. It should be noted that the effect of disabling all constraints in CP is minimal as the standard deviation in computational time is less than 10 secs across all 3 bands.

Table 16, Table 17, Table 18 show the number of IP violations in the line balance incurred by disabling the constraints for bands 1, 26, and 47 respectively. We set the maximum ergonomic score for each band as shown in the 2nd column for those experiments in which the ergonomic constraints are included. In the “Number of violations” column, the total number of mounting and assignment constraints is listed. It should be noted that it is not possible to violate all constraints at the same time, as the total number of constraints represent the number of ways a violation may occur.

Table 16: Number of IP violations in Band 1

Exp	Constraints			Max Ergonomic Score	Number of Violations	
	Ergonomics (Max=165)	Mounting	Assignment		Mounting worker interference(119)	Assignment (640)
V1.1	×	×	×	299.77	12	36
V1.2	✓	×	×	163.95	10	32
V1.3	×	✓	×	259.96	0	31
V1.4	×	×	✓	326.52	13	0
V1.5	✓	✓	×	164.56	0	27
V1.6	✓	×	✓	164.47	5	0
V1.7	×	✓	✓	215.62	0	0
V1.8	✓	✓	✓	164.47	0	0

Table 17: Number of IP violations in Band 26

Exp	Constraints			Max Ergonomic Score	Number of Violations	
	Ergonomics (Max=200)	Mounting	Assignment		Mounting worker interference(1896)	Assignment (126)
V26.1	×	×	×	385.42	67	36
V26.2	✓	×	×	199.83	51	33
V26.3	×	✓	×	-*	-	-
V26.4	×	×	✓	331.25	48	0
V26.5	✓	✓	×	-	-	-
V26.6	✓	×	✓	197.95	42	0
V26.7	×	✓	✓	403.97	0	0
V26.8	✓	✓	✓	199.98	0	0

* No optimal solution obtained.

Table 18: Number of IP violations in Band 47

Exp	Constraints			Max Ergonomic Score	Number of Violations	
	Ergonomics (Max=200)	Mounting	Assignment		Mounting worker interference(650)	Assignment (362)
V47.1	×	×	×	456.59	40	41
V47.2	✓	×	×	199.97	17	28
V47.3	×	✓	×	407.03	0	35
V47.4	×	×	✓	365.64	23	0
V47.5	✓	✓	×	-	-	-
V47.6	✓	×	✓	199.79	27	0
V47.7	×	✓	✓	418.28	0	0
V47.8	✓	✓	✓	199.49	0	0

3.5 Discussion

In this section, we discuss the sensitivity analysis results obtained for each band. The distribution of both the ergonomic score and mounting positions are shown for each band followed by a comment on the results.

3.5.1 Band 1 (50 tasks):

The histogram for the ergonomic scores of tasks in band 1 is shown in Figure 10. A high ergonomic score means the task is physically demanding and might cause injury to the worker performing it. Results for band 1 - experiment 1 are shown in Table 10. The average ergonomic score for band 1 is 24.29. The first "maximum ergonomic score" level used for this band is 155, in which an optimal solution is found with 9 workers. Increasing the maximum ergonomic score to 160 reduced the number of workers in the optimal solution to 8. The IP struggles when the maximum ergonomic score is increased to 175 as the lower bound is reduced to 7 workers but the best solution found within the time limit is 8. The CP managed to find the optimal solution in 7 out of 10 replications. The combinatorial problem of finding a combination of tasks with a total score of 175 assigned to 7 workers and conforming to the rest of constraints seems to be harder than other levels of maximum ergonomic score. The ergonomic constraint is not binding if a maximum score limit of 180 or more are used as the solution to the problem is 7 workers which is the lower bound to the problem.

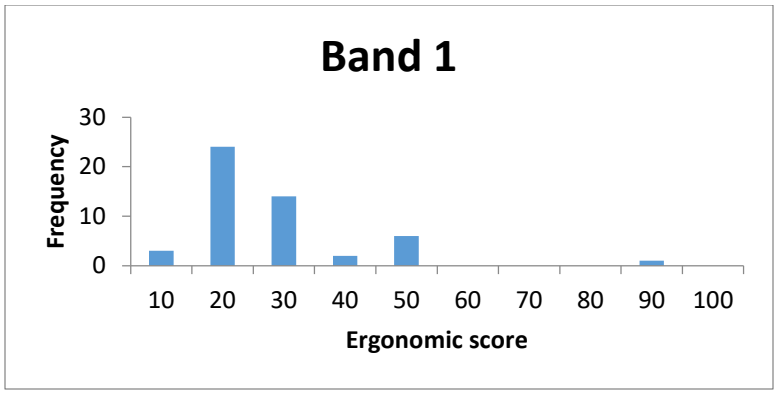

Figure 10: Band 1 ergonomic score histogram

The mounting positions used for tasks in band 1 are shown in Figure 11. Mounting position 0 is for tasks that are sub assembled on the side of the station. The tasks are distributed on the four corners of the car; this is reflected on the lower count of mounting position constraints and violations shown in Table 16. Also, this explains why this band behaved differently when the mounting position constraints are the only constraints enforced as shown in Table 13. The solution time did not change significantly as it did in the other two bands. The biggest impact on solution time occurred when both the mounting and ergonomics constraints are enforced. The IP solution time increased from a minute to over 7 hours. The CP solution time did not change and is consistent across the different instances of experiment 2.

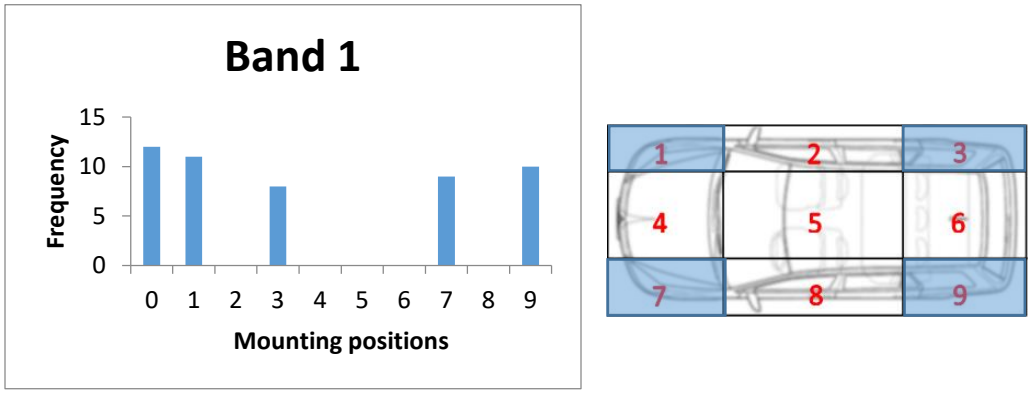

Figure 11: Band 1 mounting positions histogram

3.5.2 Band 26 (75 tasks):

Ergonomic scores of band 26 tasks are shown in the histogram of Figure 12. The average ergonomic score for band 26 is 22.53. Unlike band 1, band 26 histogram shows a trend in the relation between the number of tasks and ergonomic score. Results for band 26 - experiment 1 is shown in Table 11. Starting maximum ergonomic score for experiment 1 is 150; solving the problem gives an optimal line balance with 12 workers. Increasing the maximum ergonomic score to 170 makes the problem hard to solve for the IP. The number of workers are reduced to 11 but the lower bound is 10. The IP is unable to reach the optimal solution within the time limit. The CP managed to find an optimal line balance with 10 workers in 3 out the 10 replications. In addition, the IP was unable to reach the optimal when the maximum score limit is increased to 250 while the CP managed to get to the optimal in all 10 replications. Increasing the maximum ergonomic score to 350 or more makes this constraint a non-binding constraint.

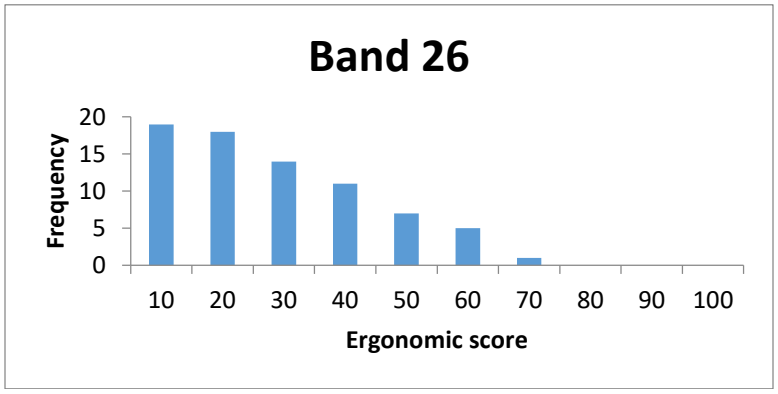

Figure 12: Band 26 ergonomic score histogram

The tasks in band 26 mostly belongs to two mounting position as shown in Figure 13. This is the reason behind having so many mounting position constraints. It took the IP 5 minutes to reach the optimal solution without enforcing any of the soft constraints. Adding the ergonomics constraints increased the time to reach the optimal for the IP. The huge number of mounting

constraints affected the IP as expected; no optimal solution is found by the IP within the time limit. By enforcing the assignment constraint only the IP solution time is reduced to less than a minute. By enforcing all soft constraints, the IP is able to find an optimal solution in 10 minutes. Disabling the mounting constraints only led to an increase in the IP's computation time. In addition, by disabling the assignment constraints, the IP is unable to reach the (relaxed) optimal solution within the time limit.

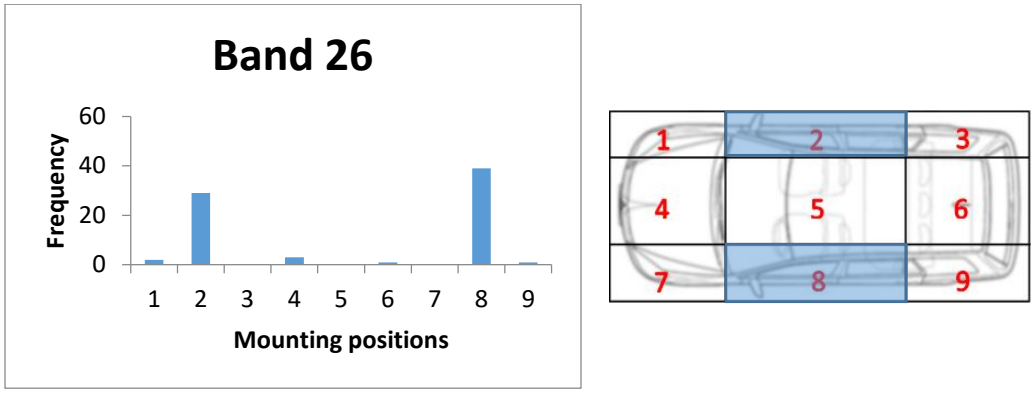

Figure 13: Band 26 mounting positions histogram

The number of band 26 violations recorded for experiment 2 is shown in Table 17. The highest number of violations occurred when no soft constraints are enforced. Among the 75 tasks, there are 67 worker interference cases in the line balancing schedule. This number might look small when compared to the number of mounting constraints, but it is not possible to violate all of the constraints at the same time and this number is large relative to the number of tasks. The number of assignments violation is also large relative to the total number of tasks. That is, almost half of the tasks violated the assignment constraint. Enforcing either the ergonomics, the assignment, or both constraints led to a decrease in mounting worker interference violations.

3.5.3 Band 47 (100 tasks)

The histogram for the ergonomic scores of band 47 is shown in Figure 14. The average ergonomic score for this band is 28.29. The scores for this band are almost uniform with more tasks having a low ergonomic score. This is reflected on the wide range of maximum ergonomic scores for which the IP is able reach an optimal solution. The results of band 47 - experiment 1 is shown in Table 12; the maximum ergonomic score for this band is varied between 90 and 500. The lower maximum ergonomic score used (90) yielded a line balance with 32 workers while the highest level used (300) resulted in a line balance with only 11 workers. This range gives the decision maker a good room to change the maximum ergonomic score based on the balance of cost/workers' health risks. Again for one level of maximum ergonomic score (110), the IP is unable to reach the optimal solution within the time limit. However, the CP reached the optimal solution in 5 out of the 10 replications. The time of CP computations is more consistent when compared to the time of IP computations.

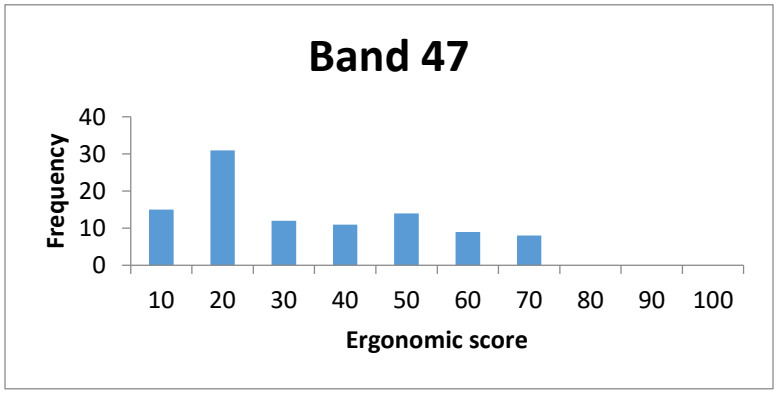

Figure 14: Band 26 ergonomic score histogram

The mounting positions used in band 47 are shown in Figure 15. In this band, all mounting position are used except for mounting position 5. The results for band 47 - experiment 2 are shown in Table 15. Solving the problem with no soft constraints enforced yielded a line balance with 11 workers. It took the IP over 12 hours to reach the optimal solution; perhaps there are many alternate

solutions, and symmetry breaking constraints would help. When the mounting position constraints are the only soft constraints enforced, the IP computation time is doubled when comparing to the solution time when no constraints are enforced. For the case when the mounting position constraints are disabled and the assignment and ergonomic constraints are enforced, the IP computation time is reduced from 12 hours to 1 hour. Also, enforcing only the assignment constraints reduced the IP time to reach the optimal solution. On the other hand, the IP is unable to reach an optimal solution within the time limit when the assignment constraints are removed and the other two soft constraints are enforced. Enabling all constraints reduced the computation time to less than half the case with no constraints enforced.

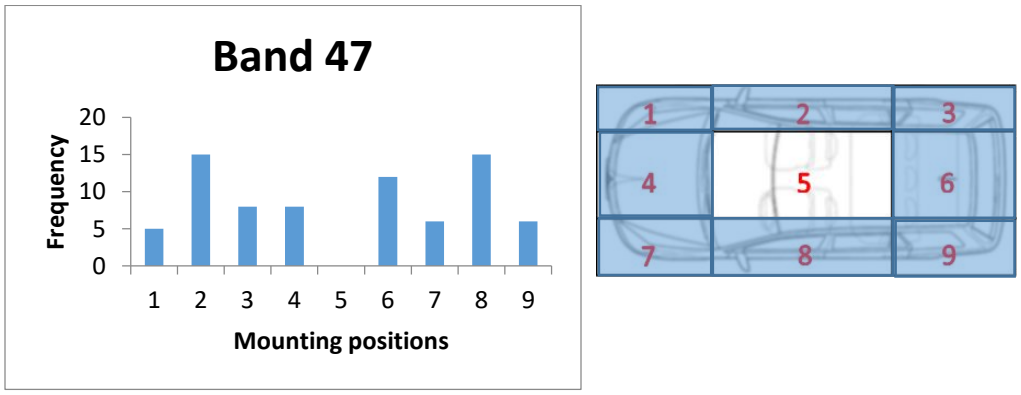

Figure 15: Band 47 mounting positions histogram

The number of constraint violations recorded for each instance of band 47 - experiment 2 is shown in Table 18. When disabling all soft constraints, the number of mounting and assignment violations amount to less than half the number of tasks on this band which is still high. In this line balance, one of the workers is assigned tasks with a total weighted ergonomic score of 456.59. By applying only the ergonomic constraints, both the mounting and assignment violations are reduced.

3.6 Conclusion

In this chapter, two sensitivity analysis experiments are conducted using data from an automaker. The first experiment studies the effect of changing the maximum ergonomic score limit on the solution quality and computational time. Results show that the problem becomes harder for some levels of maximum ergonomic level limit. For these instances, the IP failed to reach the optimal solution within the allowed time. However, the optimal solution is found in some of the CP runs for each hard instance. The decision maker should be aware of this when deciding on the maximum ergonomic limit. The second experiment studies the effect of disabling a subset of constraints on the solution quality and computational time. Results show that the assignment constraint helps the IP reach the optimal solution within the time limit. By disabling the assignment constrain the IP is unable to reach the optimal in two bands and the computation effort required to reach the optimal increased in one band. Furthermore, using only the mounting constraint led to a huge computational effort to reach the optimal in one band while the optimal is not even reached in one band.

These three bands, while from an operating OEM and hence observed and not designed for this study, provide varied characteristics that may be useful in future research. The bands have different distributions of ergonomic scores and different types of mounting positions that may provide some hypotheses for future research. For example, a further study of the impact of ergonomic score distribution can be conducted assuming all parts are in the same mounting position and only one worker can be assigned to each station; this would extend the concept presented earlier of a two-dimensional bin packing problem enhanced with precedence constraints. As a result of such a study, individual tasks could be targeted for redesign or increased attention, based on their impact on the overall line balance. As another example of a future study, we observe

that two bands studied here had disjoint mounting positions while the third had contiguous mounting positions. Further, in band 26, the mounting positions were such that no single person could work in more than one, while the potential for a person to work in more and more mounting positions was moderate in band 1 and extreme in band 47. The types of creative assignments of tasks to workers in the bands with more potential for work being done in adjacent mounting positions may merit further investigation.

Exploring other soft constraints that are just a preference and may not have significant technical implications (e.g., some precedence relations) maybe a good direction for future research. By identifying which of these constraints are hard on the IP, the computation time can be reduced without affecting the quality of the results obtained. In addition, since the results of this study is not consistent, further investigation is needed.

CHAPTER FOUR

4 MIXED MODEL SEQUENCING PROBLEM

4.1 Introduction

Mixed model assembly lines are used to produce a variety of customized products in a cost efficient flow mass production environment. Customers have the privilege of selecting the options they would like to be included in their product and the manufacturer will have to manage significant product variety as a result. In order to keep costs down, manufacturers utilize the worker/machine flexibility that is available in the mass production environment to jointly produce different unique products in an intermixed fashion on the same line. After dealing with the long term problem of line balancing, another short term problem is faced which is the sequencing problem. The sequencing problem is solved to answer the question of how to sequence the given different models for production in a production run. Different production sequences will have different economic impacts that are related to worker utilization or material usage. The sequencing problem in the literature is introduced using two different general objectives, namely work overload minimization and leveling part usage. Work overload occurs when several work intensive models are sequenced in a consecutive manner, which can be avoided by choosing a sequence in which models that require much work alternate with others that require less work. Furthermore, since different options usually require different parts or material, the model sequence is an important factor in parts logistics. In this work we will be concentrating on the work overload minimization objective. See Boysen et al. (2009a) for a survey on level scheduling research.

According to Boysen et al. (2009a), there are two different approaches in the literature to handle the work overload problem:

- The mixed model sequencing problem: In this approach, a detailed schedule of the production run is sought taking into account operation times, worker movement, station length, station borders, and other line characteristics.
- The car sequencing problem: This approach avoids the troubles of collecting data that is needed for the detailed mixed model sequencing by minimizing the work overload in an indirect manner by formulating a set of sequencing rules. These rules are to limit the occurrence of certain time consuming options in a given length of order sequence. The objective is to minimize the violations to these rules.

In the following section a review on the comparison between the two approaches is given followed by a literature review on related work. The subsequent section illustrates the problem description with an example. This is followed by model building and mathematical formulation. Next, the proposed solution approach is shown followed by computational experiments.

4.2 Mixed model sequencing versus car sequencing

The two approaches to handle the work overload problem in a mixed model assembly line are the mixed model sequencing problem (MMS) and the car sequencing (CS) problem. Although the two problems may have the same business goal, they are based on two different mathematical formulations with different objective functions. The MMS objective directly minimizes the work overload while the CS problem has an indirect objective in which the number of sequencing rules' violations is minimized. These rules are generated using different approaches that aim to avoid having several labor-intensive car models in consecutive order which in turn causes work overload. CS is a more aggregate approach for finding a production sequence when compared with MMS. It assumes that different models can be distinguished by either having the option or not. For each option o, CS uses a sequencing rule $H_o : N_o$ to limit the occurrence of that option to at most H_o in

any subsequence of N_o models. For example, a rule 2:5 for the option sunroof means that among any subsequence of five models only two of them may have the option sunroof.

Golle et al. (2014) conducted experiments to quantify the difference in solution quality between the two approaches. They used different approaches to generate the rules from the literature such as the Bolat and Yano sequencing rule which assumes only one option at each station. Since the Bolat and Yano rule is limited and was found to exclude feasible solutions from consideration, Golle et al. (2014) also tested the multiple sequencing rule approach that ensures that no feasible sequence violates a sequencing rule. In addition, the authors considered the three most commonly used objective functions in the literature. Also, several weighting factors from the literature were used. Tests were done on a large number of randomly generated instances. The results showed that although the objectives of the two approaches are positively/ linearly correlated, the MMS solution quality outperformed the CS counterpart. The authors claim that the solution of the CS problem is not even competitive across all test instances and the different objectives used. According to the authors, the CS results in solutions with at least 23% more work overload when compared to the MMS problem. They managed to decrease this gap to 15% by considering their weighting factor for CS. Furthermore, they noted that the difference in quality increases with the use of inadequate rules and objective functions. They concluded that the use of surrogate objective function in CS aggregate too much data which makes it less useful in finding a sequence that truly minimizes the work overload. Following this result, we will be using the MMS approach.

4.3 Problem description

In mixed model assembly lines, different models that have a variety of customizations are assembled on the same line. Since these models require different processing times at each station depending on the custom options required, usually some of them will require an amount of time that is higher than the cycle time while others will have a processing time which is less than the cycle time. If several of the models with the high processing times are produced in a consecutive order, workers at some station will have a hard time finishing the required task before reaching the end of the station which will result in the inability to return to the beginning of the station when a new work piece enters the station, or in other words, the worker will fall behind. This shift in a worker's position will accumulate as long as models that require a processing time that is higher than the cycle time are sequenced after each other. Work overload occurs when the worker is unable to finish working on the work piece within the boundaries of the station. If work overload occurs, several reactions are possible including but not limited to:

- The assembly line is stopped until all work is done on work pieces across all stations.
- Utility workers are used either to help workers or to take over the work on the work piece that is causing the overload.
- The unfinished work is left to be done off line in a special station after the work piece exists the last station of the line.
- The worker increases his/her production speed at the risk of quality issues.

To avoid any of the costly reactions to work overload, mixed model sequencing attempts to find a sequence such that cars with options that require a high processing time alternates with other cars that require less processing time at each station. The use of utility workers has been studied in the literature as one of the methods to handle work overload but the detail of how utility workers

are used differed. The assumption that is usually used in the literature was named the side-by-side policy by Boysen et al. (2011). In this assumption the utility worker is called whenever an overload situation is anticipated. The utility worker is assumed to support the regular worker by working with him side by side and doubling his processing speed. This way the worker will only require half the duration of the work overload amount to finish processing the required tasks on the car. It is assumed that the utility worker help starts at a position in the station such that the work required at that station is done by the right hand border of the station assuming that the flow of the assembly is from left to right. Boysen et al. (2011) argue that the assumption of side-by-side policy is restrictive and not realistic because, for example, work piece size may allow only one worker, the assumption of doubling speed is not realistic and dependent on the interaction between the workers, and utility workers cannot simply show up exactly when needed. As an alternate solution, they propose the skip policy which they claim is being used by major European car manufacturers. In the skip policy, the worker calls the utility worker whenever a work overload is expected. The utility worker then takes over the work on the work piece that is expected to cause the overload while the normal worker skips it and starts working on the next work piece coming after it. The utility worker is usually a group leader who is responsible for several stations and who is cross trained to be able to handle all tasks that are assigned to any of his stations. The difference between the two policies is illustrated with an example in the following section.

Stations on a production line might be closed in the sense that no operators can cross their boundaries to an adjacent station. This limitation is necessary if the working area require specific environmental conditions such as heating champers or paint shops. On the contrary, workers can cross borders of open stations but the range of how far they can go into an adjacent stations is always limited. The restriction might be related to a limit on the range of a power tool or the

window of the material handling system. Furthermore, the areas of adjacent stations may overlap but workers from the two stations are not allowed to interfere with each other by working on the same work-piece at the same time. Open stations are useful in mixed model production because it possible for a worker who finished working early on a work-piece to “swim” (move upstream) to the previous station so he can have more time to work on labor-intensive models. With a proper production sequence, the worker can avoid any work overload or the need of the utility worker help.

4.4 Related work

Yano & Rachamadugu (1991) investigate the mixed model sequencing problem with the objective of minimizing work overload. First, the authors propose an optimal algorithm for the case of one station and two products. Then they propose a greedy heuristic that is based on branch and bound to solve the problem with multiple stations. They tested the proposed heuristic on data provided by an automobile company and compared the heuristic results against the results the company get using their own procedure. Yano & Rachamadugu (1991) claim that the proposed heuristic reduces total work overload by 55% on average.

Bolat & Yano (1992a) develop a procedure to minimize utility work for the MMS problem with one station, two types of products, and no buffers. They prove that the proposed spacing rule simple procedure is optimal if the load in the system is sufficiently high or low. In addition, they develop error bounds on the algorithm if the load conditions are not satisfied. For a moderately loaded system, Bolat & Yano (1992a) develop a greedy algorithm that is optimal under certain processing time requirements. For cases in which the optimality is not guaranteed, the authors develop a dynamic program in addition to a variation of the greedy algorithm proposed. Results show that the greedy algorithm outperforms the spacing rule procedure over some parameter

values. The authors note that in the class of problems for which the solutions are still far from optimal require a combination of different procedure to be solved. Moreover, Bolat & Yano (1992b) introduce a surrogate objective to the utility work that yields a scheduling problem that is easier to solve. The objective is a car sequencing rule of not having a number of cars with an option in any sequence of a given length.

Tsai (1995) considers the problem of mixed model sequencing for one station with the objectives of minimizing both utility work and the risk of line stoppage. The author lists two different management philosophies in handling work overload in U.S. and Japan. In the U.S. the utility worker is deployed to finish work left undone by the primary worker. On the other hand in Japan the operator pushes a stop button when he is unable to finish his work. A proof that the problem of minimizing either objective is NP hard in the strong sense is given first. Assuming that product processing times can take only one of two distinct values, Tsai (1995) proposes an optimal algorithm that minimizes both objectives in O(log N) computation time. Furthermore, the objective value from the single station case can be used as a lower bound for a multiple stations/processing times case.

Sarker & Pan (1998) consider a mixed model assembly line with both open and closed stations. They study the problem of minimizing the utility and idle time costs that are incurred due to various line parameters along with the sequencing of different models. The authors develop two different models for the open-station and closed-station systems in order to find the optimal parameters and sequences for each system. The models are able to provide the best parameters that would minimize the total cost. The results show that for a given line length, the open-station system incurred less cost when compared to the closed-station counterpart.

Scholl et al. (1998) study the MMS problem with fixed launch rate, closed stations, and the objective of minimizing the work overload. They propose a reformulation of the problem to consider patterns in subsequences that can be put together to make the complete sequence. The basic idea is to decompose the problem of finding the complete sequence to problems of finding appropriate patterns and then combining them to get the solution. They define a pattern to be a subsequence in which the worker resets his position to the station border after completing all the work required for that subsequence. Scholl et al. present a method to solve the problem by first generating patterns based on a pattern based vocabulary building strategy using the column generation technique and then searching for a complete sequence (solution) using tabu search. Furthermore, the authors show that the vocabulary building procedure of the method is not affected by the number of products to be sequenced while other phases of the method will be affected by the computational complexity of solving for a longer sequence.

Bautista & Cano (2008) propose a single station sequencing heuristic algorithm with two types of products that is based on the heuristics proposed by Yano & Rachamadugu (1991) and Bolat & Yano (1992a). They also extend their heuristic to account for multiple products and multiple stations. Results show a satisfactory performance in terms of quality and speed.

Cano-Belmán et al. (2010) use a hyper heuristic to solve the problem of MMS with closed stations (without links). The proposed hyper heuristic uses scatter search as high level metaheuristic along with a low level priority rule constructive heuristic. The authors compare the two proposed hyper heuristics against other algorithms from Bautista and Cano (2008), improved best solution using local search, and exact solution using CPLEX. They find that improvement methods are needed to get good quality solutions that outperform other heuristics in comparison.

Bautista & Cano (2011) extend the models of Yano and Rachamadugu (1991) with the objective of maximizing the total work completed in the assembly line, and Scholl (1998) with the objective of minimizing the work overload. The proposed models take into consideration the link between neighboring stations. This is done by adding a new constraint that limits the starting position of the work done on a work piece form starting while work is not finished in the previous station. The variables of the model are based on the work piece instead of a period. The authors introduce an interruption rule for the management of interruptions and work intensification and propose two new models based on the rule to make the stations more autonomous. Furthermore, they propose a solution that they named the bounded dynamic programming to solve the given sequencing problem. Computational results on 225 instances form the literature show the fast performance of the proposed method compared to the CPLEX solver. Next, the authors present a case study from the Nissan powertrain plant in Barcelona to test the new proposed solution method on real world assembly lines. The solution and lower bounds of the proposed approach were better compared to CPLEX but results were not guaranteed to be optimal.

Bautista et al. (2012) present two models for mixed model sequencing problem for assembly lines with stations in series and homogeneous processors in parallel extending the models in Bautista & Cano (2011). The difference between the two models is the use of absolute time scale versus the use of relative time scale. The same assumptions that the line can be stopped and operation interruption can happen anytime while retaining the work in progress still hold. In addition, it is assumed that a worker who did not complete all the work at the station is allowed to leave the unit so that the unfinished work can be completed later. Bautista et al. (2012) define an alternate surrogate for workload minimization which is minimizing the variation of work overload rates across all stations on the line. They propose different approaches to avoid work overload

concentrations at each workstation and at specific time instants of the work day. These approaches are regulating the work required, the work completed, or the overload at a station throughout the workday. They apply the proposed approaches to a set of examples representing different scenarios from the powertrain production line of a Nissan plant in Barcelona. Finally, they compare the different approaches and conclude that models with regularity functions produce better results when compared to previous models.

Boysen et al. (2011) study utility work organization in the MMS problem with closed stations. They propose a more realistic problem in which the objective is to minimize the work overload situations. In the literature, the side-by-side policy is often assumed in which a utility worker helps the regular worker finish the remaining work whenever the regular worker is unable to finish the work assigned to him within the stations' boundaries. In the proposed policy, a normal worker skips working on a work-piece whenever work overload is expected and the utility worker starts working alone on that work-piece (calling this the skip policy). In addition, Boysen et al. (2011) develop a branch and bound procedure along with different heuristics to solve the problem. Finally, the authors show the advantage of the proposed skip policy by comparing it to the traditional side-by-side policy.

Tamura et al. (2011) consider the MMS problem with the objective of minimizing the line stoppage time. They propose a formulation for the problem and derive a relationship between the line stoppage time and makespan. The authors develop a branch and bound algorithm and show the limitation of solving numerical examples using commercial mathematical programming packages. They conclude that heuristics are needed to solve larger problems.

Gujjula et al. (2011) develop a heuristic that is based on Vogel's approximation to solve the MMS problem. The motivation for their work is to find a method to solve real-life large

problems. They use two other heuristics from the literature to benchmark the results of the test scenarios; the first is a combination of a priority rule-based method with lower bounds determination, and the second is an adjusted version of the heuristic proposed by Bautista and Cano (2008). Finally, Gujjula et al. (2011) show the performance of the proposed heuristic by comparing the results to the results of the benchmark heuristics.

Cortez, & Costa (2015) study the problem of MMS with a heterogeneous workforce which is motivated by assembly lines in sheltered work centers for the disabled. Based on heterogeneous workforce, task processing time is both model and worker dependent. The authors formulate the problem and propose a constructive heuristic method that is based on a simplified and approximated models for the problem. Furthermore, Cortez, & Costa (2015) propose a meta-heuristic that is based on the greedy randomized adaptive search procedure (GRASP). The authors show the speed of proposed heuristics by comparing it against the solution obtained using CPLEX.

Bautista et al. (2015) consider the mixed model sequencing in which the objective is to minimize the work overload. They propose to consider the work pace factor in the model by the use of variable processing times with the objective of minimizing the work overload or increasing the completed work. The activity of workers is set by using a function that represents the periods of adaptation and fatigue throughout the working day. The authors use a case study from the Nissan powertrain plant in Barcelona to analyze the performance of the proposed model using the Gurobi solver. The results show that a very small increase in the activity of the operators can reduce the work overload or completely avoid it.

Recently, Mosadegh et al. (2016) study the problem of MMS with one station, stochastic processing times, and the objective of minimizing the expected total work overload. The authors find that their proposed dynamic programming approach is not guaranteed to find the optimal

solution because of the recursive computation required to calculate the work overload. To overcome this problem, the authors reformulated the problem as a shortest path problem. They propose a heuristic that is based on Dijkstra's algorithm to solve the problem and show the quality of the solution obtained by comparing to results from other methods.

4.4.1 An example

Consider an assembly line with 3 stations, in which each station has a length of 13 time units (TU) which is the time a work-piece spends in the station. Assume that every 10 TU a new work-piece is launched down the assembly line (cycle time) and that workers are not allowed to swim past the station borders. Furthermore, assume that the processing time for each model at each station is as shown in Table 19 and the demand is 2 units of models 1 & 2 and 1 unit of models 3 & 4. Let the sequence be (model 1, model 1, model 2, model 2, model 3, and model 4).

Table 19: Example data for the mixed model sequencing problem

Model	**Demand**	**Time required at Station 1**	**Time required at Station 2**	**Time required at Station 3**
1	2	13	8	11
2	2	10	12	13
3	1	8	10	9
4	1	9	11	10

In the figures, the production sequence starts from the bottom and ends at the top of the figure. Figure 16 shows the position and movement of both the regular and utility workers for the side-by-side policy. The horizontal solid blocks represent the movement of the regular worker when he is working on a work-piece while the diamond filled blocks represent the movement of both the regular and utility workers when they are working together on a work-piece. The dashed diagonal lines represent the return movement of the regular worker to start working on the next work-piece.

The outlined diamond blocks represent the time in which both the utility worker and the normal worker are both working side-by-side on the work piece. The time it takes the regular worker to reposition is assumed to be instantaneous.

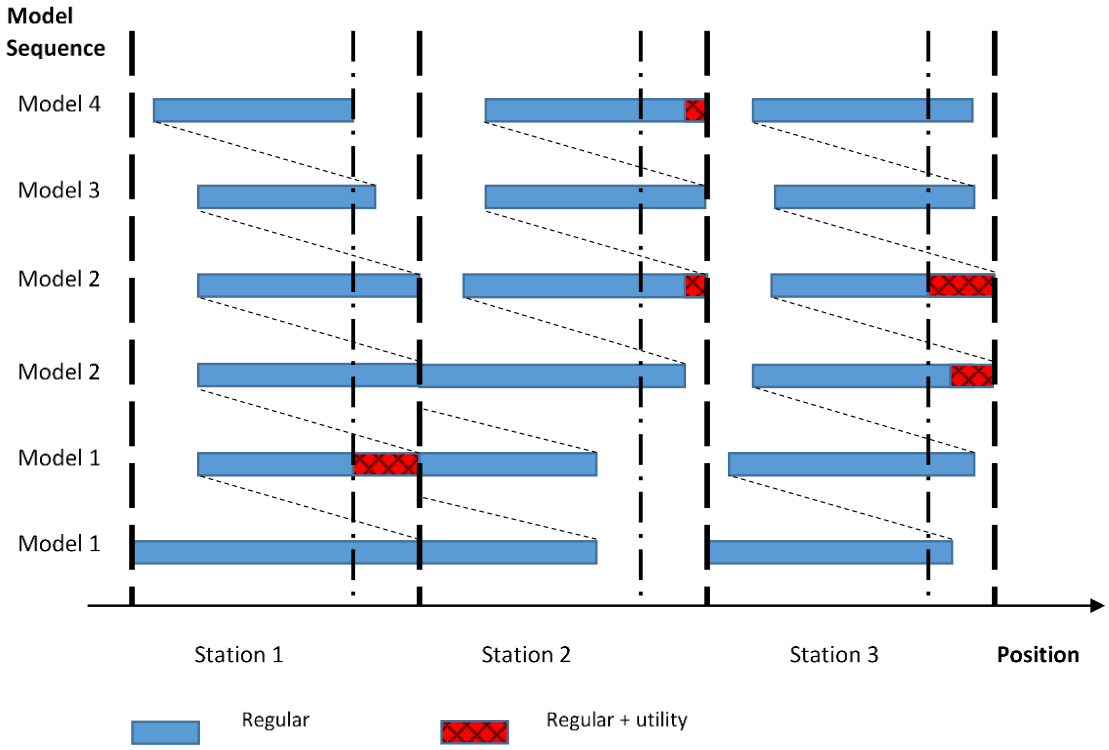

Figure 16: Movement in the side-by-side policy

Figure 17 shows the movement for the skip policy case, resulting in an overload usage count of 3 compared to 5 in the side by side policy for the same given sequence. The checkerboard blocks represent the time the utility worker is working at the regular pace alone on the work-piece.

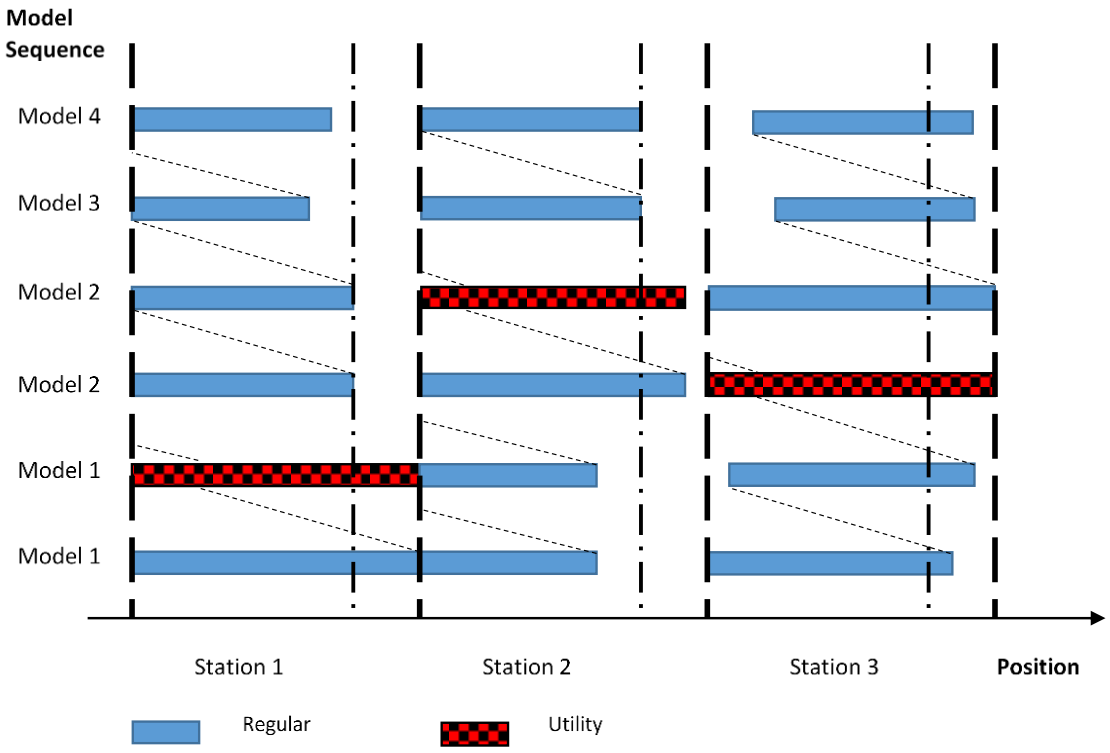

Figure 17: Movement in the skip policy

If it is assumed that a regular worker can swim across his station's border to the previous station, he can start working on the coming work piece in the station that precedes his station if he is ready and if the worker in the previous station has finished the required tasks on that work piece early. The number of overload usage can be reduced for the given sequence if swimming is allowed as in Figure 18.

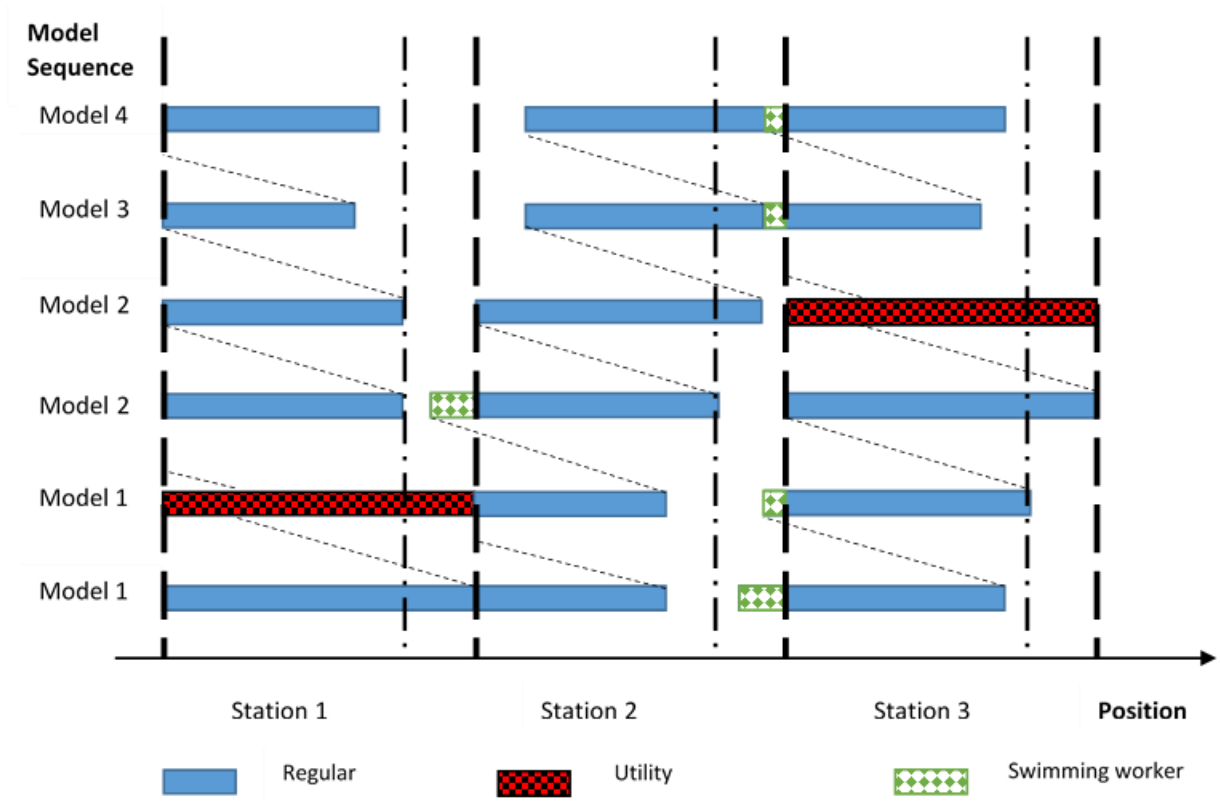

Figure 18: Movement with swimming allowed

Even if swimming of workers is not allowed, an optimal model sequence to the example reduce the number of overload usage count to 1. The detailed worker movement diagram for an optimal solution is shown in Figure 19. The worker in station 2 working on the first Model 2 car is able to swim into the previous station to start working on it early. This way, no utility worker is needed for the second Model 2 car at station 2 because the overload situation is avoided.

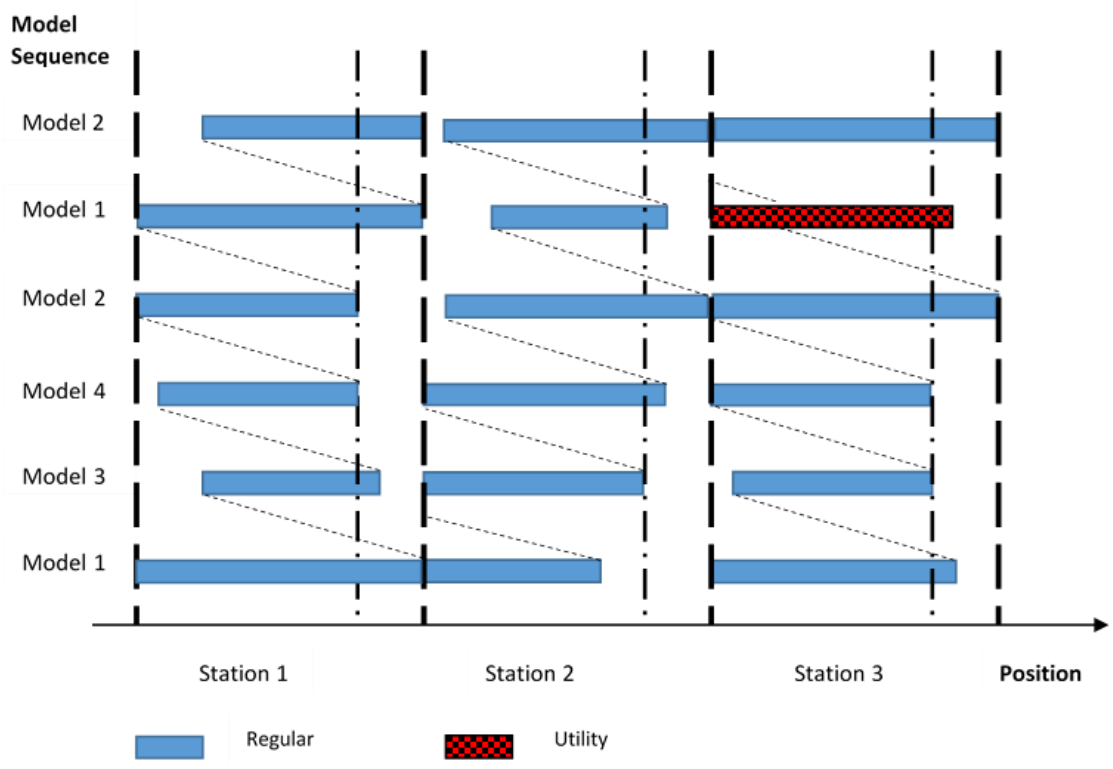

Figure 19: Worker movement in an optimal solution for the example

In this work, the skip policy with swimming allowed is used in the formulation and the solution of the mixed model sequencing problem with the objective of minimizing the work overload situations. The following assumptions are often used in real world assembly lines and are used to model the problem:

- The mixed model demand and the processing time of all tasks is deterministic.
- Working across the borders of the station is possible (open stations), and we assume that a worker can swim (drift) to the previous station so he can start working on the work piece earlier if the worker in that station finished working before the end of the station.
- The processing time for all tasks is always less than the station's length.

- Regular workers will not be required to drift to the next station to complete any task since utility workers take care of overload situations and tasks are always less than the station length.
- The length units (LU) and time units (TU) are the same assuming that conveyor velocity is normalized to 1 LU/TU.
- The operator returns to the next work piece with infinite velocity. This is because the speed of the conveyor is usually much slower than the human speed.
- Regeneration is not required, i.e. there is no need to reset workers' position to zero after the last period.
- Stations may have different lengths and different starting positions for workers.
- The cycle time is fixed and work pieces are launched down the line in equal intervals.

4.5 Model formulation

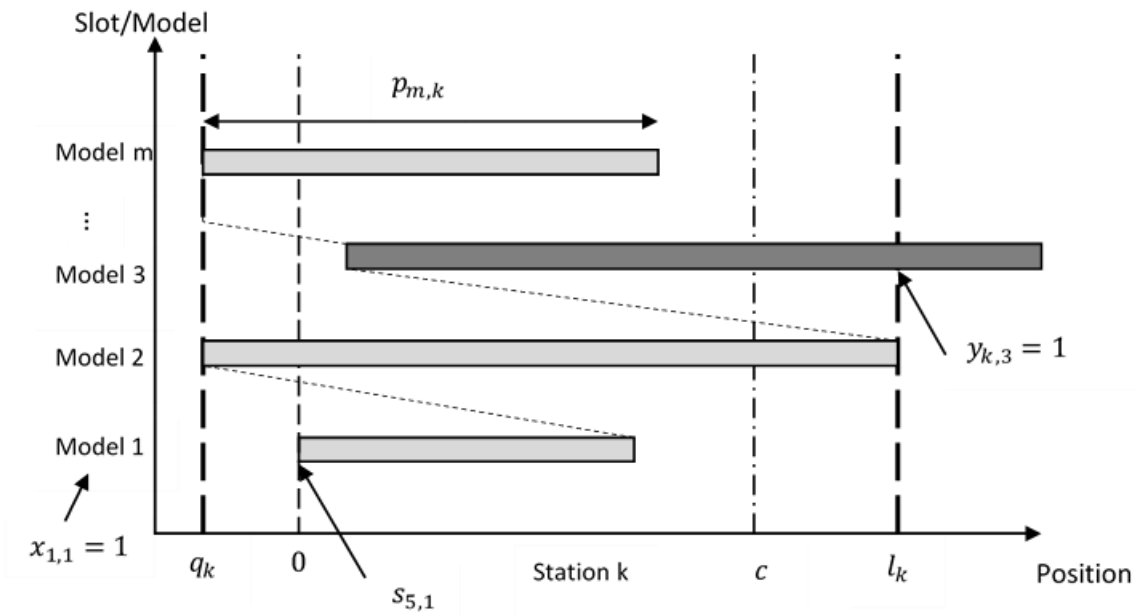

Figure 20: Movement of a worker in a station

In this work, we extend the model provided by Boysen et al. (2011) by adding station to station interaction via worker swimming. We propose an IP and a heuristic to solve the MMS problem with swimming and assess their performance using data from Bautista & Cano (2011). Therefore, we will compare the IP and heuristic performance as opposed to comparing our heuristic results to Boysen et al. (2011) heuristic results.

The parameters and variables needed for the MMS are shown in Table 20 and are illustrated in Figure 20. Each station k has a length l_k that is greater than the cycle time c, and an early start position q_k that may be less than 0 to allow the worker to swim to the previous station so he can start working early on the next work piece in sequence if that is needed. The worker will not be able to swim to the previous station unless the worker in that station finishes his work early

in the previous period (due to the potential for work location overlap). The binary decision variable x_{mt} takes the value 1 when car model m is assigned to period t and 0 otherwise.

If the projected ending time for model m at station k $(p_{m,k})$ in period t is greater than the station's length $(s_{k,t} + \sum_m p_{mk} x_{mt} \geq l_k)$, a utility worker is called to help and the regular worker completely skips the overload causing work piece to start working on the work piece that follows it $(y_{k,t} = 1)$. Thus, the starting position of each worker depends on both the finishing position of the same worker in the previous period and the finishing position of the worker in the previous station in the previous period. This can be described as:

$$s_{k,t} = \begin{cases} \max\left(f_{k,t-1} - c, f_{k-1,t-1} - l_{k-1}, q_k\right) & if\ y_{k,t-1} = 0 \\ \max\left(s_{k,t-1} - c, f_{k-1,t-1} - l_{k-1}, q_k\right) & if\ y_{k,t-1} = 1 \end{cases} \tag{4.1}$$

Where $f_{k,t} = s_{k,t} + \sum_m p_{mk} x_{mt}$

4.6 Mathematical model

Table 20: Mixed model sequencing problem's notations

M	Number of models indexed $m = 1,2,3, \dots, M$
T	Number of production cycles indexed $t = 1,2,3, \dots, T$
K	Number of stations indexed $k = 1,2,3, \dots, K$
c	Cycle time
l_k	Length of station k
p_{mk}	Processing time of model m at station k
d_m	Demand for copies of model m ($d_m = 1$ if all cars are unique)
q_k	Minimum starting position for the worker at station k ($q_k = 0$ if no swimming is allowed)
a_k	Initial starting position for the worker at station k
ϵ	A very small number

Decision variables:

$$x_{mt} = \begin{cases} 1 & if\ model\ m\ is\ assigned\ to\ cycle\ t \\ 0 & otherwise \end{cases}$$

$$y_{kt} = \begin{cases} 1 & if\ utility\ worker\ is\ required\ in\ station\ k\ cycle\ t \\ 0 & otherwise \end{cases}$$

s_{kt} = Starting position of worker in station k when cycle t begins.

The model may be represented with the use of an additional variable f_{kt} that represents the finishing position of worker in station k when cycle t ends. In the following we use the expression $s_{kt} + \sum_m p_{mk} x_{mt}$ instead of introducing an additional decision variable to the integer program.

Objective:

$$minimize\ Z = \sum_t \sum_k y_{kt}$$

The objective is to minimize the number of overload situations across all production cycles and stations.

Subject to the following constraints:

- Only one model is assigned to each production cycle.

$$\sum_{m} x_{mt} = 1 \qquad \forall\, t = 1, \dots, T \qquad \text{(C4.1)}$$

- Demand for car models should be satisfied during the production horizon.

$$\sum_{t} x_{mt} = d_m \qquad \forall\, m = 1, \dots, M \qquad \text{(C4.2)}$$

- The variable that indicates an overload situation is flagged using the following constraints. This variable y_{kt} should be equal to 1 if an overload will occur when the finishing position of the worker f_{kt} exceeds the station's length l_k. (Recall, $s_{kt} + \sum_m p_{mk} x_{mt}$ represents the finishing position of worker in station k when cycle t ends). It should be noted that a redundant constraint is used (C4.4) compared to the formulation in Boysen et al. (2011) in which only one constraint is used. The reason is because when one constraint only is used, y_{kt} will be free to take any value in some cases but will be pushed to equal zero since the minimization of the sum of y_{kt} is the objective. The redundant constraint will help setting the flag variable y_{kt} correctly regardless of the objective function and will define the feasible region more completely.

$$s_{kt} + \sum_{m} p_{mk} x_{mt} - l_k - \epsilon \geq -l_k(1 - y_{kt}) \qquad \forall\, t = 1, \dots, T,\ k = 1, \dots, K \qquad \text{(C4.3)}$$

$$s_{kt} + \sum_{m} p_{mk} x_{mt} - l_k \leq l_k y_{kt} \qquad \forall\, t = 1, \dots, T,\ k = 1, \dots, K \qquad \text{(C4.4)}$$

- The starting position of the worker of each station depends on the finishing position of the same worker in the previous cycle if no utility worker is needed (no overload situation). In

the case of overload situation, a utility worker is called to help and the worker can skip the current cycle's car and start working on the next cycle's car.

$$s_{kt} \geq s_{k(t-1)} + \sum_{m} p_{mk} x_{m(t-1)} - c - l_k y_{k(t-1)} \qquad \forall\, t = 2, \dots, T,\ k = 1, \dots, K \tag{C4.5}$$

$$s_{kt} \geq s_{k(t-1)} - c - l_k + l_k y_{k(t-1)} \qquad \forall\, t = 2, \dots, T,\ k = 1, \dots, K \tag{C4.6}$$

- A worker cannot start working on a car unless the worker in the previous station has finished working on it. In case the previous station had a utility worker in the previous period, the utility worker will always finish working on the car before the end of the station.

$$s_{kt} \geq s_{(k-1)(t-1)} + \sum_{m} p_{m(k-1)} x_{m(t-1)} - l_{k-1} - (l_{k-1} - c) y_{(k-1)(t-1)} \qquad \forall\, t = 2, \dots, T,\ k = 2, \dots, K \tag{C4.7}$$

- The starting position of workers should be greater than or equal to the minimum starting position allowed in his station and less than or equal the difference between the length of the station and the cycle time.

$$s_{kt} \geq q_k \qquad \forall\, t = 1, \dots, T,\ k = 1, \dots, K \tag{C4.8}$$

$$s_{kt} \leq l_k - c \qquad \forall\, t = 1, \dots, T,\ k = 1, \dots, K \tag{C4.9}$$

- Initial starting positions for workers at the first cycle.

$$s_{k1} \geq a_k \qquad \forall\, k = 1, \dots, K \tag{C4.10}$$

- Variable domains.

$$x_{mt} \in \{0,1\} \qquad \forall\, t = 1, \dots, T, m = 1, \dots, M \tag{C4.11}$$

$$y_{kt} \in \{0,1\} \qquad \forall\, t = 1, \dots, T,\ k = 1, \dots, K \tag{C4.12}$$

4.7 Solution approach

Since the problem of mixed-model sequencing (MMS) is an NP-hard optimization problem, we expect that there will be limitation in using the Integer Program (IP) to solve realistically sized problems in a reasonable amount of time. However, it is useful to have an IP that is able to solve small and medium sized problems and that can be used as a benchmark to any other solution methods used. In this section we present a meta-heuristic named the greedy randomized adaptive search procedures (GRASP) to solve the problem at hand.

4.7.1 Greedy randomized adaptive search procedures (GRASP)

Metaheuristics represent a family of approximate optimization general and high-level procedures that coordinate simple heuristics or rules to find good quality solution for hard combinatorial optimization problem in reasonable time.

The GRASP metaheuristic is an iterative greedy heuristic that was proposed by Feo and Resende (1989) as a heuristic to solve the set covering problem and later was named in the following paper by Feo and Resende (1995). GRASP is composed of two steps: construction of a solution and local search. In the construction step, the solution is constructed by iteratively adding elements to the partial solution in a greedy way such that each added element has a minimal impact on the solution's objective value. After a solution is constructed, local search is performed to check the solution's neighborhood for a better solution. The greedy algorithm is randomized to be able to generate several solutions. Since the local search depends on the initial solution, this metaheuristic will be efficient if the constructive heuristic samples different promising parts of the search space.

4.7.1.1 Constructive greedy randomized algorithm

Consider the combinatorial optimization problem of minimizing $f(S)$ over all solutions $S \in X$, which is defined by a set of feasible solutions X, and by an objective function f. The first step in the GRASP metaheuristic is to generate a good feasible solution to the problem. The heuristic starts with an empty solution and appends an element from a restricted candidate list RCL to the solution at each iteration. The determination of the next element to be added to the solution is done in part using a greedy evaluation function that represents the incremental increase in cost that accompanies adding that element to the partial solution. Using that function a list of candidate elements that have the lowest increase on the cost of the partial solution is generated. The candidate set is the set of all possible elements that can be incorporated into the solution. Furthermore, the RCL is the list of candidate elements that will impact the objective the least. In order to make use of randomness in generating different solutions with good quality, the element to be added to the partial solution is selected randomly from the RCL. A pseudo code for the greedy randomized solution generation is given in Figure 21.

```
Algorithm: Constructive greedy randomized algorithm.
1.  S ← ∅;
2.  Populate the candidate set C;
3.  Evaluate the incremental cost c(e) for all e ∈ C;
4.  while C ≠ ∅
5.  Build a restricted candidate list with elements that have an
incremental cost equal to the minimum across the candidate set.
RCL ← e  ∀ c(e) = min(c(e): e ∈ C);
6.      Select element s ∈ RCL at random;
7.      Incorporate s into the solution:  S ← S ∪ {s};
8.      Update the candidate set C;
9.      Reevaluate the incremental cost c(e) for all e ∈ C;
10.  end
11. return S;
end
```

Figure 21: Greedy randomized algorithm for a minimization problem

4.7.1.2 Neighborhood function

The definition of the neighborhood function is a required step in implementing any metaheuristic that is based on a single solution (such as simulated annealing and tabu search) and not a population of solutions (such as genetic algorithms). This function should be selected carefully because it plays an important role in the performance of the metaheuristic. A neighborhood of a solution S is defined by the set $N(S) \subseteq X$. The members of the neighborhood set are generated by the application of a move operator that performs a small perturbation to the solution S. It is desirable that a neighborhood solution be local by performing a move operation that yields small changes in the solution value. There are two general types of operators for a permutation solution representation. The first is the position-based neighborhood such as the insertion operator in which one random element is removed from its position to be inserted into a new random position in the sequence. The insertion operation is illustrated in Figure 22. The

second is the order-based neighborhood, with operators like the exchange and inversion operators illustrated in Figure 23 and Figure 24 respectively. In the exchange operator, two randomly selected elements are swapped while in the inversion operator, two random elements are selected and the sequence between these elements is inverted.

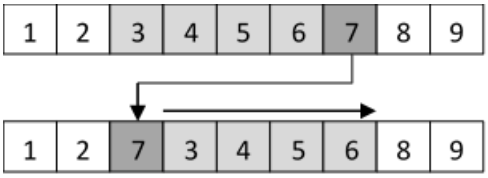

Figure 22: Insertion operator

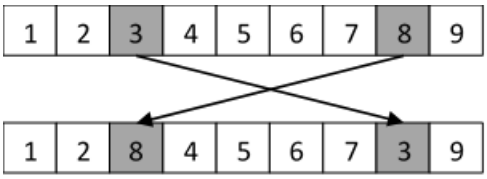

Figure 23: Exchange (swap) operator

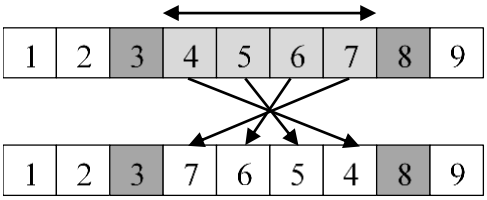

Figure 24: Inversion operator

4.7.1.3 Local search

Local search iteratively generates solutions from a predefined neighborhood $N(S)$ and moves to a better solution once and if it is found. The exploration of the neighborhood is usually stochastic because enumerating the whole neighborhood is time-prohibitive. A pseudo code example of a local search is given in Figure 25.

```
Algorithm: Local search
while the max number of iteration is not reached do
    Find S' ∈ N(S) with f(S') < f(S);
    S ← S';
end
return S;
end
```

Figure 25: Local search algorithm

4.7.1.4 GRASP for the MMS problem:

There are two primary design decisions to implement GRASP for the MMS problem: the objective function and the neighborhood.

Objective function:

For a given sequence, let $b_{k,t}$ denote the current processing time for the model assembled in station k in cycle t. In order to know the number of overload situations for this given sequence, the starting $s(k,t)$ and finishing $f(k,t)$ position of workers at all stations $k = 1, \dots, K$ for all production sequences $t = 1, \dots, T$ should be calculated. The overload for each station at each production cycle $o(k,t)$ can be easily calculated using the skip policy detailed earlier. The number of overload situation for the given sequence is equal to the sum of all overload situation across all the stations and production cycles. The algorithm for the evaluation of a given sequence is given in Figure 26.

```
Algorithm: overload situations for a given sequence
1. Initialize starting positions at t=1
   for k = 1 to K
        s(k,1) = 0;
   end
2. Calculate workers' finish & start positions for all production cycles and record any overload situation
  for  k = 1 to K
        f(k,1) = s(k,1) + b(k,1);
   end
  for t = 2 to T
        if f(1,t-1) ≤ l(1) & o(1,t-1) == 0
              s(1,t) = max{0, f(1,t-1) - c};
        else
              s(1,t) = max{0, s(1,t-1) - c};
end
f(1,t) = s(1,t) + b(1,t);
              if f(1,t) ≥ l(1)
                    o(1,t) = 1;
                    s(1,t) = 0;
                    f(1,t) = s(1,t) + b(1,t);
              end
 end
  for  k = 2 to K
        for t = 2 to T
              if f(k,t-1) ≤ l(k) & o(k,t-1) == 0
                    s(k,t) = max{q(k), f(k,t-1) - c, f(k-1,t-1) - l(k-1)};
              else
                    s(k,t) = max{q(k), s(k,t-1) - c, f(k-1,t-1) - l(k-1)};
              end
              f(k,t) = s(k,t) + b(k,t);
              if f(k,t) ≥ l(k)
                    o(k,t) = 1;
                    s(k,t) = 0;
                    f(k,t) = s(k,t) + b(k,t);
              end
        end
end
5. Calculate the total overload
  Totaloverload = Σ_k Σ_t o(k,t);
```

Figure 26: Computing the objective function by evaluating a sequence

4.7.1.5 Neighborhood function for the MMS problem:

The solution to the MMS problem is naturally represented by a permutation since it is the models' sequence to be assembled on the assembly line. The adjacency of the models in the sequence affects the number of overload situations. Thus, the order-based neighborhood is more suitable for the MMS problem and specifically the exchange (swap) operator. This operator swaps the position of a predefined number of elements in the sequence. Swapping more than two elements in the sequence of the MMS problem makes a big perturbation that might never lead to a better solution. For the GRASP algorithm, the 2- exchange operator is used in which two production orders in the sequence are swapped randomly.

4.7.1.6 The constructive randomized greedy algorithm:

The first step in the randomized greedy algorithm is to populate the candidate set, C, which in our case is the different models to be assembled on the assembly line. The inputs to the algorithm are the task duration on all stations for all models, cycle time, stations' length, swimming amount allowed for each station, and the demand of each model. Initially, the candidate list will have all models and the solution can start by picking a model randomly. However if swimming is not allowed, the task with highest total task duration across all stations can be picked first since it is the most demanding model and the starting position for workers are initialized to zero. The next step in the greedy algorithm is to update the candidate list, by updating the demand for each model and removing models with a demand of zero, and calculate the cost (overload situations) of adding each of the models to the sequence. After that, a restricted candidate list (RCL) is generated from the candidate list by keeping models that will incur the minimum cost when added to the sequence and removing all the other candidate models. Afterwards, one of the models in the RCL is selected randomly and appended to the sequence. The procedure will iterate until all demand is satisfied.

4.8 Computational experiments

In our computational experiments, we use instances from the Nissan powertrain plant in Barcelona that are presented in Bautista & Cano (2011). The plant assembles nine types of engines that are grouped under three families (4x4, vans and trucks) through an assembly line that is composed of 21 workstations. The stations are arranged serially and are linked to each other. In addition, each station has the same number of operators under normal operating conditions. The processing time for each type of engine on each stations is provided in Table 21. This table is a result of a line balance presented in Bautista and Pereira (2007) that is done for the Nissan powertrain plant in Barcelona. The 46 instances are grouped into two blocks: the first has 23 demand plans each for 270 units, which is equivalent to a production schedule for a full working day with two shifts. The second block has 23 demand plans for 540 units, which is equivalent to a production schedule for two working days. An effective cycle time (c=175s) is used in the experiments with a fixed station length for all stations (l=195s) with no swimming allowed (q=0). The demand plans for the 46 instances are shown in Table 22.

Table 21: Processing time for the 9 types of engines

i	Stations																				
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**
p1	104	103	165	166	111	126	97	100	179	178	161	96	99	147	163	163	173	176	162	164	177
p2	100	103	156	175	114	121	96	97	174	172	152	106	101	155	152	185	179	167	150	161	161
p3	97	105	164	172	114	122	96	95	173	172	168	105	102	142	156	183	178	181	152	157	154
p4	92	107	161	167	115	124	93	106	178	177	167	97	101	154	152	178	169	180	152	159	168
p5	100	101	148	168	117	127	96	94	178	178	167	101	99	146	153	169	173	172	160	162	172
p6	94	108	156	167	117	130	89	102	171	177	166	100	101	143	152	173	178	173	151	160	170
p7	103	106	154	168	115	120	94	103	177	175	172	96	96	154	154	172	174	173	155	162	167
p8	109	102	164	156	111	121	101	102	171	173	157	104	102	153	156	182	175	168	148	158	149
p9	101	110	155	173	111	134	92	100	174	175	177	96	99	155	156	171	175	184	167	157	169

Table 22: Demand plans for 46 instances

	i	Block 1																						
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
4x4	**p1**	30	30	10	40	40	50	20	20	70	10	10	24	37	37	24	30	30	60	10	20	60	20	10
	p2	30	30	10	40	40	50	20	20	70	10	10	23	37	37	23	30	30	60	10	20	60	20	10
	p3	30	30	10	40	40	50	20	20	70	10	10	23	36	36	23	30	30	60	10	20	60	20	10
Van	**P4**	30	45	60	15	60	30	75	30	15	105	15	45	35	45	55	35	55	30	90	15	15	90	30
	p5	30	45	60	15	60	30	75	30	15	105	15	45	35	45	55	35	55	30	90	15	15	90	30
Trucks	**p6**	30	23	30	30	8	15	15	38	8	8	53	28	23	18	23	28	18	8	15	45	15	8	45
	p7	30	23	30	30	8	15	15	38	8	8	53	28	23	18	23	28	18	8	15	45	15	8	45
	p8	30	22	30	30	7	15	15	37	7	7	52	27	22	17	22	27	17	7	15	45	15	7	45
	p9	30	22	30	30	7	15	15	37	7	7	52	27	22	17	22	27	17	7	15	45	15	7	45
	Total	270	270	270	270	270	270	270	270	270	270	270	270	270	270	270	270	270	270	270	270	270	270	270
	i	**Block 2**																						
		24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46
4x4	**p1**	60	60	20	80	80	100	40	40	140	20	20	47	74	74	47	60	60	120	20	40	120	40	20
	p2	60	60	20	80	80	100	40	40	140	20	20	47	73	73	47	60	60	120	20	40	120	40	20
	p3	60	60	20	80	80	100	40	40	140	20	20	46	73	73	46	60	60	120	20	40	120	40	20
Van	**P4**	60	90	120	30	120	60	150	60	30	210	30	90	70	90	110	70	110	60	180	30	30	180	60
	p5	60	90	120	30	120	60	150	60	30	210	30	90	70	90	110	70	110	60	180	30	30	180	60
Trucks	**p6**	60	45	60	60	15	30	30	75	15	15	105	55	45	35	45	55	35	15	30	90	30	15	90
	p7	60	45	60	60	15	30	30	75	15	15	105	55	45	35	45	55	35	15	30	90	30	15	90
	p8	60	45	60	60	15	30	30	75	15	15	105	55	45	35	45	55	35	15	30	90	30	15	90
	p9	60	45	60	60	15	30	30	75	15	15	105	55	45	35	45	55	35	15	30	90	30	15	90
	Total	540	540	540	540	540	540	540	540	540	540	540	540	540	540	540	540	540	540	540	540	540	540	540

The integer program experiments were run on AMPL version 20161220 using two different versions of the Gurobi solver, namely Gurobi 6.5.0 and Gurobi 7.0.2 solvers. The GRASP experiments were coded and run on MATLAB R2016a. Block 1 and Block 2 experiment with 50,000 iterations were run on a machine that has an Intel core i7-3770 @3.4 Ghz processor, 32GB RAM, and Windows 7. Block 2 experiments with 100,000 iterations were executed on a machine with an Intel core i7-2600 @ 3.4 Ghz processor, 16GB RAM, and Windows 7.

IP results using the two versions of the Gurobi solver are shown in Table 23. The lower bound (LB) and best solution (number of overload situations) are reported along with the computation time required to reach the best solution. Computational time might be very small compared to the 3600 seconds time limit because the branch and bound method used by the IP solver might get stuck on a solution early on and need over 3600 seconds to find a better solution. The time column is the time at which the reported solution was found. The newer version of the solver found better solutions in 17 out of the 46 instances, while it found worse solutions in 14 out of the 46 instances (best values bolded). Furthermore, for instances in which both versions found the same objective function value, the newer version was faster in 6 instances and slower in 6 (lower time bolded). Also, the newer version found a better lower bound in 3 instances but had a worse lower bound in 1. The better result, regardless of which solver found it, is used in the rest of this chapter.

Table 23: IP results for two versions of Gurobi

instance		Gurobi ver 6.5.0 (3600 Seconds time limit)			Gurobi ver 7.0.2(3600 Seconds time limit)		
		LB	**Solution**	**Time* (sec)**	**LB**	**Solution**	**Time* (sec)**
Block 1	1	3	8	2856	3	**7****	324
	2	11	**20**	1929	11	22	2135
	3	18	24	**582**	18	24	1611
	4	9	13	3558	9	13	**2501**
	5	23	**34**	3203	**24**	36	3443
	6	11	21	**1292**	11	21	2858
	7	30	38	**203**	30	38	2823
	8	4	4	3111	4	4	**1680**
	9	25	36	1178	25	**34**	3122
	10	51	60	1810	51	**58**	474
	11	3	3	**281**	3	3	360
	12	8	17	2921	**10**	17	1908
	13	7	**14**	23	7	15	223
	14	**13**	26	1129	12	**25**	2441
	15	17	**24**	1681	17	28	217
	16	5	13	**168**	5	13	479
	17	18	29	267	18	**28**	186
	18	18	29	1730	18	**28**	109
	19	40	**47**	571	40	49	67
	20	3	3	3322	3	3	**2283**
	21	18	30	1703	18	**27**	2140
	22	41	**49**	2174	41	50	951
	23	5	5	17	5	5	**9**
Block 2	24	6	26	800	6	**20**	1939
	25	23	44	2905	23	**42**	1006
	26	38	**50**	801	38	53	465
	27	20	34	1197	20	**33**	2004
	28	50	84	41	50	**77**	1125
	29	24	**49**	2602	24	54	2392
	30	63	**81**	2710	63	84	602
	31	7	**12**	2817	7	15	909
	32	52	80	3397	52	**77**	927
	33	103	120	1261	103	**116**	3479
	34	6	7	1492	6	7	**1229**
	35	21	**36**	2745	21	46	810
	36	16	38	1607	16	**36**	878
	37	27	55	959	27	55	**183**
	38	35	52	1516	35	52	**1346**
	39	11	28	3547	11	**27**	2309
	40	37	**53**	2521	37	62	1022
	41	36	75	185	**37**	**66**	1449
	42	81	100	3595	81	**97**	731
	43	6	13	**2903**	6	13	3530
	44	38	**65**	852	38	67	1789
	45	84	**105**	1920	84	110	484
	46	10	11	3598	10	11	**2350**

* The time at which the reported solution was found.

** Bolded is better.

In order to benchmark the GRASP algorithm in solving the MMS problem, the results obtained using the algorithm are compared against the best IP solution from Table 23. Since GRASP is based on randomization, a number of replications is done for each instance to better assess the performance of the algorithm. Each instance is solved 30 times using GRASP and the following is recorded: minimum solution, maximum solution, solution average, solution standard deviation, number of times the best solution is reached, and the average computation time. The GRASP was limited to 100,000 iterations for block 1 instances, which is equivalent to approximately 10 minutes of computation time. For block 2 instances, the 50,000 iterations is equivalent to approximately 10 minutes of computation time while the 100,000 iterations takes approximately 20 minutes to finish.

Results of the IP and GRASP algorithm for Block 1 and Block 2 are shown in Table 24 and Table 25 respectively. Note that the IP was able to find the optimal solution for only 4 instances of block 1 within the one hour time limit. On the other hand, no optimal solutions were found for block 2, which is expected due to the size of the problem. The GRASP algorithm managed to find better or optimal solutions to all instances including the ones in which the IP has failed to reach the optimal solution. In addition, the worst GRASP solution found among the 30 replications is still better than the IP solution for all but two of these instances. The average time for the GRASP to reach the best solution is around 4.4 minutes for block 1 instances, 6.2 minutes for block 2 instances with 50,000 iterations, and 14.7 minutes for block 2 instances with 100,000 iterations. Looking at Table 24 it is clear that some instances are easier than the others for the IP and GRASP. Instances 8, 11, 20, 23 in which the IP has found an optimal solution are easier than the other instances even for the GRASP algorithm. Another way to look at the difficulty of the different

instances is by looking at the number of times the best solution is found by the GRASP algorithm across the 30 replications. This number varies between instances but it is clear that the number of times the best solution is reached for instances in block 1 is higher than the same number for instances of block 2. This is expected as block 2 instances are bigger and require more computation time for the algorithm to be efficient. Furthermore, the average solution is improved when the number of iterations is doubled as we can see in Table 25. It should be noted that the instances in which the demand patterns with more truck engines (p6, p7, p8, p9) are easier to solve when compared to other instances and have lower bounds from the IP for the total number of overload situations. This is a result of how the line balance distributed the truck engines' related tasks across the assembly line stations.

Table 24: IP vs GRASP Block 1 results

Instance		IP (3600 seconds time limit)			GRASP (600 seconds time limit, 30 replications)					
		LB	Solution	Time* (sec)	Min	Max	Avg.	SD	# of best	Avg. Time* (sec)
Block 1	1	3	7	324	5	6	5.53	0.51	14	321.89
	2	11	20	1929	15	17	15.40	0.56	19	294.85
	3	18	24	582	21	22	21.27	0.45	22	184.20
	4	9	13	2501	12	13	12.50	0.51	15	236.68
	5	24	34	3203	29	32	30.37	0.81	4	344.70
	6	11	21	1292	15	18	17.10	0.84	1	389.64
	7	30	38	203	34	35	34.17	0.38	25	238.29
	8**	**4**	**4**	**1680**	**4**	5	4.07	0.25	28	204.29
	9	25	34	3122	30	33	31.60	0.86	3	325.19
	10	51	58	474	55	56	55.17	0.38	25	234.60
	11	**3**	**3**	**281**	**3**	**3**	3.00	0.00	30	37.22
	12	10	17	1908	12	13	12.53	0.51	14	332.26
	13	7	14	23	10	13	11.50	0.63	1	315.72
	14	13	25	2441	17	19	18.13	0.78	7	336.19
	15	17	24	1681	20	22	20.20	0.48	25	291.06
	16	5	13	168	8	10	8.57	0.63	15	322.96
	17	18	28	186	21	24	22.33	0.71	3	363.42
	18	18	28	109	23	26	24.37	0.67	1	354.98
	19	40	47	571	43	45	43.80	0.55	8	221.81
	20	**3**	**3**	**2283**	**3**	5	3.87	0.68	9	233.42
	21	18	27	2140	23	25	24.17	0.65	4	307.45
	22	41	49	2174	45	46	45.47	0.51	16	186.23
	23	**5**	**5**	**9**	**5**	**5**	5.00	0.00	30	63.56

* The time at which the reported solution was found.
** Bold indicates that an optimal solution was found.

Table 25: IP vs GRASP Block 2 results

Instances		IP (3600 seconds time limit)			GRASP (50,000 iterations, 30 replications)						GRASP (100,000 iterations, 30 replications)					
		LB	Solution	Time* (sec)	Min	Max	Avg.	SD	# of best	Avg. Time* (sec)	Min	Max	Avg.	SD	# of best	Avg. Time* (sec)
Block 2	24	6	20	1939	15	19	16.73	1.01	2	440.90	13	17	15.13	1.01	1	840.63
	25	23	42	1006	33	39	36.63	1.27	1	410.05	34	37	34.97	0.85	9	965.92
	26	38	50	801	43	46	44.90	0.76	1	317.87	43	46	44.33	0.71	3	722.22
	27	20	33	2004	27	31	28.77	0.97	2	451.55	26	29	27.63	0.85	3	919.57
	28	50	77	1125	65	69	67.27	1.20	4	429.57	64	67	65.97	0.89	1	954.46
	29	24	49	2602	38	44	41.17	1.49	1	441.83	37	41	38.67	1.06	5	1055.59
	30	63	81	2710	71	74	72.40	0.81	3	415.26	70	73	71.37	0.72	2	861.42
	31	7	12	2817	9	13	11.30	0.84	1	353.50	8	12	10.23	1.01	2	957.99
	32	52	77	927	69	72	70.23	1.04	8	425.39	66	70	67.93	1.36	6	1080.62
	33	103	116	3479	112	115	113.90	0.80	1	261.68	112	114	113.27	0.58	2	668.76
	34	6	7	1229	7	8	7.10	0.31	27	252.69	7	7	7.00	0.00	30	413.72
	35	21	36	2745	28	33	30.63	1.19	1	408.86	27	30	28.83	0.91	1	991.19
	36	16	36	878	27	32	29.60	1.28	1	394.73	26	30	27.87	0.97	2	925.01
	37	27	55	183	41	45	43.17	1.09	2	394.87	39	45	41.77	1.14	1	1025.49
	38	35	52	1346	43	47	44.87	1.11	2	376.74	42	46	43.53	1.04	4	940.22
	39	11	27	2309	21	25	22.70	0.99	3	426.92	19	23	20.80	1.00	2	1056.14
	40	37	53	2521	47	53	50.77	1.41	1	410.32	48	51	49.23	0.82	4	975.48
	41	37	66	1449	52	59	56.27	1.55	1	404.08	51	56	53.77	1.22	2	1056.06
	42	81	97	731	90	91	90.73	0.45	8	254.97	89	91	90.27	0.52	1	670.71
	43	6	13	2903	9	13	10.73	0.98	3	375.46	8	11	9.97	0.56	1	819.02
	44	38	65	852	53	57	54.67	1.35	7	427.03	51	54	52.37	1.03	7	1023.20
	45	84	105	1920	93	96	94.83	0.65	1	356.54	92	95	94.03	0.76	2	868.90
	46	10	11	2350	11	12	11.67	0.48	10	226.12	11	12	11.20	0.41	24	609.56

In order to assess the benefit of swimming on minimizing the overload situation, experiments are conducted using GRASP to compare the results when $q_k = 0$ and $q_k = -17$ for $k = 2, .., .K$. The swimming amount q_k is chosen to be 10% of the cycle time $c = 175$. The experiments are done with the same random number seed for both cases. Table 26 and Table 27 show the results for blocks 1 and 2 respectively. Results show that by allowing workers to swim 10% of the cycle time the work overload situations is reduced by 46.6% and 48% on average for blocks 1 and 2 respectively. Swimming had most impact on instance 16 of block 1 in which the average of work overload situations is reduced by %58.8 from 9.7 to 4, and in instance 24 of block 2 in which the average of work overload situations is reduced by %63.2 from 14.5 to 5.3.

Table 26: Effect of swimming on Block 1 overload situations

Instances		Overload situations with no swimming, $q_k = 0$					Overload situations with swimming allowed, $q_k = -17$				
		Min	Max	Avg	SD	# of best	Min	Max	Avg	SD	# of best
Block 1	1	5	7	6.30	0.65	3	2	3	2.70	0.47	9
	2	14	18	15.93	0.94	2	7	9	8.27	0.52	1
	3	21	22	21.57	0.50	13	12	12	12.00	0.00	30
	4	12	14	12.93	0.58	6	7	8	7.50	0.51	15
	5	29	33	31.30	0.84	1	16	18	17.27	0.52	1
	6	16	19	18.30	0.65	1	8	10	9.00	0.45	3
	7	34	35	34.47	0.51	16	19	20	19.20	0.41	24
	8	4	6	4.23	0.50	24	2	2	2.00	0.00	30
	9	31	34	32.37	0.72	2	18	19	18.93	0.25	2
	10	55	56	55.67	0.48	10	32	33	32.13	0.35	26
	11	3	3	3.00	0.00	30	2	2	2.00	0.00	30
	12	12	15	13.57	0.97	5	7	7	7.00	0.00	30
	13	11	13	12.23	0.68	4	5	6	5.97	0.18	1
	14	17	20	18.83	0.79	1	9	11	9.47	0.57	17
	15	20	22	20.60	0.67	15	11	11	11.00	0.00	30
	16	8	11	9.70	0.70	1	4	4	4.00	0.00	30
	17	22	24	23.33	0.61	2	12	12	12.00	0.00	30
	18	24	27	25.40	0.67	2	13	14	13.40	0.50	18
	19	43	45	44.33	0.55	1	25	25	25.00	0.00	30
	20	3	5	4.13	0.51	2	2	2	2.00	0.00	30
	21	23	26	24.67	0.76	2	13	15	14.07	0.37	1
	22	44	47	45.63	0.61	1	26	27	26.50	0.51	15
	23	5	5	5.00	0.00	30	3	3	3.00	0.00	30

Table 27: Effect of swimming on Block 2 overload situations

Instances		Overload situations with no swimming, $q_k = 0$					Overload situations with swimming allowed, $q_k = -17$				
		Min	Max	Avg	SD	# of best	Min	Max	Avg	SD	# of best
Block 2	24	13	16	14.50	0.94	5	5	6	5.33	0.48	20
	25	33	37	35.07	1.23	2	16	18	16.63	0.72	15
	26	43	46	44.43	0.82	4	24	25	24.93	0.25	2
	27	26	29	27.27	0.83	5	14	16	15.03	0.72	7
	28	63	68	65.70	1.24	1	34	36	35.03	0.76	8
	29	37	41	39.27	1.01	1	19	21	19.70	0.53	10
	30	71	73	71.67	0.61	12	40	42	40.87	0.43	5
	31	9	12	10.57	0.82	3	5	5	5.00	0.00	30
	32	65	70	67.73	1.17	1	37	40	38.47	0.63	1
	33	112	115	113.30	0.65	2	65	67	65.83	0.53	7
	34	7	8	7.03	0.18	29	4	4	4.00	0.00	30
	35	27	31	28.90	0.92	2	14	14	14.00	0.00	30
	36	25	29	27.50	1.07	1	12	14	13.10	0.71	6
	37	39	44	41.37	1.10	2	19	22	20.73	0.58	1
	38	42	46	43.80	0.85	1	23	24	23.03	0.18	29
	39	19	23	21.03	1.03	1	8	11	8.83	0.79	11
	40	48	51	49.40	0.89	4	24	26	25.43	0.73	4
	41	51	55	53.93	1.01	1	27	30	28.37	0.72	3
	42	90	91	90.20	0.41	24	52	52	52.00	0.00	30
	43	9	12	10.00	0.83	9	5	6	5.10	0.31	27
	44	50	54	52.37	1.16	1	28	30	29.27	0.74	5
	45	93	95	94.13	0.57	3	54	55	54.17	0.38	25
	46	10	12	11.07	0.37	1	6	6	6.00	0.00	30

4.9 Conclusion

In this chapter, the MMS problem with open stations and skip policy is studied. The problem is introduced first and then compared to the car sequencing problem. This is followed by a literature review on related work. In this work, we adopt the skip policy in managing work overload in which a utility worker takes over all the work required for a work piece in a given station whenever an overload is expected. In the case a utility worker is called, the normal worker skips the current work piece to the next one in sequence. In addition, interaction between stations is allowed through workers swimming beyond stations boundaries. The problem description is given and both the skip policy and swimming is illustrated by examples. Next the mathematical model formulation is given followed by the details of the proposed GRASP algorithm that is developed to solve the problem.

To assess the performance of the IP and GRASP, computation experiments are done on data from the literature. Two different versions of the IP solver are used and the best result is compared against the GRASP result. Results show that the GRASP algorithm performed better in a fraction of the time when compared to the IP solution. In addition, results show that work overload situations can be reduced by up to 63.3% by allowing workers to swim a distance (LU) equal to 10% of the cycle time (TU).

The MMS problem discussed in this work and in the literature consider the movement of one worker per station. A possible future research is considering the movement of multiple workers per station along with the mounting position they are working on. For example, a worker can still swim to work on a free mounting zone while

other workers are still working on other mounting zones. This will open up the possibility of having more room for swimming. This could also change the way line balancing is done by trying to schedule tasks on certain mounting zones early to anticipate for workers swimming from the following station.

CHAPTER FIVE

5 CONCLUSION AND FUTURE WORK

In this dissertation, the model by Becker and Scholl (2009) is extended by considering the suggestions, in Falkenauer (2005) and Pearce (2015), to build a realistic mixed model line balancing model with parallel stations, zoning constraints, assignment restrictions, and ergonomics. The IP formulation is shown followed by a scheduling CP model that is developed to solve industrial sized assembly line balancing problems. Both models were compared using data provided from an OEM. Results show that the CP model outperformed the IP model and is considered a potential tool for solving the mixed model assembly line balancing problem.

In the following chapter, the OEM data are used for two sensitivity analysis experiments. The first experiment studies the effect of changing the maximum ergonomic score limit on the performance of both the IP and CP models. The second experiment examines the effect of disabling a subset of constraints on the performance of the IP and CP. Results show that CP computation time was always consistent except in some hard instances which required more computation time. The problem becomes harder to solve for some levels of maximum ergonomic score. Results also show that disabling the assignment constraint increased the time needed for the IP to reach the optimal solution.

Finally, the mixed model sequencing problem is studied. Interaction between stations are allowed through workers swimming. The skip policy is used to manage the work overload situations. An IP formulation is presented along with a GRASP algorithm that is developed to solve the problem. Results show that the GRASP algorithm managed

to find better solutions in less computation time than the IP. By allowing workers to swim, work overload situations can be reduced by up to 63.3%. This managerial insight is a key contribution of this chapter.

Future work can be based on either a microscopic or macroscopic view of the assembly systems modeled in this dissertation. A microscopic view explores the details to increase the realism of the models presented. This would make the models more applicable but would add to the complexity of the problem. One possible future direction would be to add constraints to the IP or CP models such as a cognitive constraint. Since each assembly task requires some skill and knowledge from the worker to perform it, would it be more realistic to assume that each worker has a limited knowledge of how different assembly tasks are performed? Moreover, could the demand of switching tasks, as in a station or work assignment that manages many different models, be cognitively tiring, even if it is not ergonomically or physically challenging? Another extension to the line balancing problem would be to consider a better ergonomics model instead of using a simple constraint with an upper bound. Is it possible to implement a more realistic ergonomics model in the CP and how would that affect solution time and quality? Furthermore, since tasks duration and demand values are assumed to be deterministic in this work, what if they are assumed to be stochastic? In addition, the effect of line balancing on the robustness of the sequencing problem can be studied. How would the model/task correlation in two consecutive stations affect the sequencing? Also, since line balancing is done using parallel workers; how would the mixed model sequencing problem differ if the movement of more than one worker is considered?

A macroscopic extension looks at the big picture and not the details. This approach ignores the details for the sake of solving an integrated problem. An interesting research direction would be to integrate the mixed model line balancing problem with the sequencing problem. Is it better to solve the line balancing and sequencing problems in one step? Another research idea would be to develop a model that can solve bigger problems. Is it possible to come up with a line balance for the whole assembly line? Finally, line balancing and sequencing problems can be also integrated with the incoming and outgoing supply chains. How would integrating supply chain affect the line balancing and sequencing decisions?

REFERENCES

Akpınar, S., & Bayhan, G. M. (2011). A hybrid genetic algorithm for mixed model assembly line balancing problem with parallel workstations and zoning constraints. *Engineering Applications of Artificial Intelligence*, 24(3), 449-457.

AkpıNar, S., Bayhan, G. M., & Baykasoglu, A. (2013). Hybridizing ant colony optimization via genetic algorithm for mixed-model assembly line balancing problem with sequence dependent setup times between tasks. *Applied Soft Computing*, 13(1), 574-589.

Bautista, J., & Cano, J. (2008). Minimizing work overload in mixed-model assembly lines. *International Journal of Production Economics*, 112(1), 177-191.

Bautista, J., & Cano, A. (2011). Solving mixed model sequencing problem in assembly lines with serial workstations with work overload minimisation and interruption rules. *European Journal of Operational Research*, 210(3), 495-513.

Bautista, J., Cano, A., & Alfaro, R. (2012). Modeling and solving a variant of the mixed-model sequencing problem with work overload minimisation and regularity constraints. An application in Nissan's Barcelona Plant. *Expert systems with applications*, 39(12), 11001-11010.

Bautista, J., Alfaro, R., & Batalla, C. (2015). Modeling and solving the mixed-model sequencing problem to improve productivity. *International Journal of Production Economics*, 161, 83-95.

Bautista, J., & Pereira, J. (2007). Ant algorithms for a time and space constrained assembly line balancing problem. *European journal of operational research*, 177(3), 2016-2032.

Baybars, I. (1986). A survey of exact algorithms for the simple assembly line balancing problem. *Management science*, *32*(8), 909-932.

Becker, C., & Scholl, A. (2006). A survey on problems and methods in generalized assembly line balancing. *European journal of operational research*, *168*(3), 694-715.

Becker, C., & Scholl, A. (2009). Balancing assembly lines with variable parallel workplaces: Problem definition and effective solution procedure. *European Journal of Operational Research*, 199(2), 359-374.

Bock, S. (2008). Using distributed search methods for balancing mixed-model assembly lines in the automotive industry. *Or Spectrum*, 30(3), 551-578.

Bockmayr, A., & Pisaruk, N. (2001). Solving assembly line balancing problems by combining IP and CP. *Proceedings of the 6th Annual Workshop of the ERCIM Working Group on Constraints*, Prague, Czech Republic.

Bolat, A., & Yano, C. A. (1992a). Scheduling algorithms to minimize utility work at a single station on a paced assembly line. *Production Planning & Control*, 3(4), 393-405.

Bolat, A., & Yano, C. A. (1992b). A surrogate objective for utility work in paced assembly lines. *Production Planning & Control*, 3(4), 406-412.

Boysen, N., Fliedner, M., & Scholl, A. (2007). A classification of assembly line balancing problems. *European Journal of Operational Research, 183*(2), 674-693.

Boysen, N., & Fliedner, M. (2008). A versatile algorithm for assembly line balancing. *European Journal of Operational Research, 184*(1), 39-56.

Boysen, N., Fliedner, M., & Scholl, A. (2008). Assembly line balancing: which model to use when? *International Journal of Production Economics, 111*(2), 509-528.

Boysen, N., Fliedner, M., & Scholl, A. (2009). Assembly line balancing: Joint precedence graphs under high product variety. *IIE Transactions, 41*(3), 183-193.

Boysen, N., Fliedner, M., & Scholl, A. (2009a). Sequencing mixed-model assembly lines: Survey, classification and model critique. *European Journal of Operational Research, 192*(2), 349-373.

Boysen, N., Kiel, M., & Scholl, A. (2011). Sequencing mixed-model assembly lines to minimise the number of work overload situations. *International Journal of Production Research, 49*(16), 4735-4760.

Bukchin, J., Dar-El, E. M., & Rubinovitz, J. (2002). Mixed model assembly line design in a make-to-order environment. *Computers & Industrial Engineering, 41*(4), 405-421.

Bukchin, Y., & Rabinowitch, I. (2006). A branch-and-bound based solution approach for the mixed-model assembly line-balancing problem for minimizing stations and task duplication costs. *European Journal of Operational Research, 174*(1), 492-508.

Cano-Belmán, J., Ríos-Mercado, R. Z., & Bautista, J. (2010). A scatter search based hyper-heuristic for sequencing a mixed-model assembly line. *Journal of Heuristics*, 16(6), 749-770.

Cortez, P. M., & Costa, A. M. (2015). Sequencing mixed-model assembly lines operating with a heterogeneous workforce. *International Journal of Production Research*, 53(11), 3419-3432.

Chutima, P., & Chimklai, P. (2012). Multi-objective two-sided mixed-model assembly line balancing using particle swarm optimisation with negative knowledge. *Computers & Industrial Engineering, 62*(1), 39-55.

Ege, Y., Azizoglu, M., & Ozdemirel, N. E. (2009). Assembly line balancing with station paralleling. *Computers & Industrial Engineering, 57*(4), 1218-1225.

Erel, E., & Gokcen, H. (1999). Shortest-route formulation of mixed-model assembly line balancing problem. *European Journal of Operational Research, 116*(1), 194-204.

Falkenauer, E. (2005, July). Line balancing in the real world. *In Proceedings of the International Conference on Product Lifecycle Management PLM* (Vol. 5, pp. 360-370).

Falkenauer, E. (1998) *Genetic Algorithms and Grouping Problems*. New York: John Wiley & Sons.

Feo, T. A., & Resende, M. G. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 8(2), 67-71.

Feo, T. A., & Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2), 109-133.

Gökcen, H., & Erel, E. (1997). A goal programming approach to mixed-model assembly line balancing problem. *International Journal of Production Economics, 48*(2), 177-185.

Gökcen, H., & Erel, E. (1998). Binary integer formulation for mixed-model assembly line balancing problem. *Computers & Industrial Engineering, 34*(2), 451-461.

Golle, U., Rothlauf, F., & Boysen, N. (2014). Car sequencing versus mixed-model sequencing: A computational study. *European Journal of Operational Research*, 237(1), 50-61.

Gujjula, R., Werk, S., & Günther, H. O. (2011). A heuristic based on Vogel's approximation method for sequencing mixed-model assembly lines. *International Journal of Production Research, 49*(21), 6451-6468.

Haq, A. N., Rengarajan, K., & Jayaprakash, J. (2006). A hybrid genetic algorithm approach to mixed-model assembly line balancing. *The International Journal of Advanced Manufacturing Technology, 28*(3-4), 337-341.

Hop, N. V. (2006). A heuristic solution for fuzzy mixed-model line balancing problem. *European Journal of Operational Research, 168*(3), 798-810.

Jin, M., Luo, Y., & Eksioglu, S. D. (2008). Integration of production sequencing and outbound logistics in the automotive industry. *International Journal of Production Economics, 113*(2), 766-774.

Kara, Y., Özgüven, C., Seçme, N. Y., & Chang, C. T. (2011). Multi-objective approaches to balance mixed-model assembly lines for model mixes having precedence conflicts and duplicable common tasks. The *International Journal of Advanced Manufacturing Technology, 52*(5-8), 725-737.

Mahdavi, I., Javadi, B., Sahebjamnia, N., & Mahdavi-Amiri, N. (2009). A two-phase linear programming methodology for fuzzy multi-objective mixed-model assembly line problem. *The International Journal of Advanced Manufacturing Technology, 44*(9-10), 1010-1023.

Manavizadeh, N., Hosseini, N. S., Rabbani, M., & Jolai, F. (2013). A Simulated Annealing algorithm for a mixed model assembly U-line balancing type-I problem considering human efficiency and Just-In-Time approach. *Computers & Industrial Engineering, 64*(2), 669-685.

Matanachai, S., & Yano, C. A. (2001). Balancing mixed-model assembly lines to reduce work overload. *IIE Transactions*, 33(1), 29-42.

McMullen, P. R. (1998). JIT sequencing for mixed-model assembly lines with setups using tabu search. *Production Planning & Control, 9*(5), 504-510.

McMullen, P. R., & Frazier, G. V. (1997). A heuristic for solving mixed-model line balancing problems with stochastic task durations and parallel stations. *International Journal of Production Economics, 51*(3), 177-190.

McMullen, P. R., & Tarasewich, P. (2003). Using ant techniques to solve the assembly line balancing problem. *IIE transactions, 35*(7), 605-617.

McMullen, P. R., & Tarasewich, P. (2006). Multi-objective assembly line balancing via a modified ant colony optimization technique. *International Journal of Production Research, 44*(1), 27-42.

Mosadegh, H., Fatemi Ghomi, S. M. T., & Süer, G. A. (2016). Heuristic approaches for mixed-model sequencing problem with stochastic processing times. *International Journal of Production Research*, 1-24.

Otto, A., & Scholl, A. (2011). Incorporating ergonomic risks into assembly line balancing. *European Journal of Operational Research*, 212(2), 277-286.

Özcan, U., Toklu, B. (2009). Balancing of mixed-model two-sided assembly lines. *Computers & Industrial Engineering, 57*(1), 217-227.

Öztürk, C., Tunalı, S., Hnich, B., Örnek, A. (2013). Balancing and scheduling of flexible mixed model assembly lines with parallel stations. *The International Journal of Advanced Manufacturing Technology, 67*(9-12), 2577-2591.

Öztürk, C., Tunalı, S., Hnich, B., & Örnek, M. A. (2013b). Balancing and scheduling of flexible mixed model assembly lines. *Constraints*, 18(3), 434-469.

Pastor, R., Ferrer, L., & García, A. (2007). Evaluating optimization models to solve SALBP. In International Conference on Computational Science and Its Applications (pp. 791-803). Springer Berlin Heidelberg.

Pearce, B. (2015). *A study on general assembly line balancing modeling methods and techniques.* (Doctoral dissertation). Retrieved from Clemson University's Dissertation database (tigerprints No. 1549).

Pil, F.K., & Holweg, M. (2004). Linking product variety to order-fulfilment strategies. *Interfaces, 34*, 394–403.

Sarker, B. R., & Pan, H. (1998). Designing a mixed-model assembly line to minimize the costs of idle and utility times. *Computers & industrial engineering*, 34(3), 609-628.

Sawik, T. (2002). Monolithic vs. hierarchical balancing and scheduling of a flexible assembly line. *European Journal of Operational Research, 143*(1), 115-124.

Scholl, A., Klein, R., & Domschke, W. (1998). Pattern based vocabulary building for effectively sequencing mixed-model assembly lines. *Journal of Heuristics*, 4(4), 359-381.

Scholl, A. (1999). *Balancing and sequencing assembly lines* (2nd ed.). Physica, Heidelberg.

Scholl, A., & Becker, C. (2006). State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research, 168*(3), 666-693.

Sewell, E. C., & Jacobson, S. H. (2012). A branch, bound, and remember algorithm for the simple assembly line balancing problem. *INFORMS Journal on Computing, 24*(3), 433-442.

Simaria, A. S., & Vilarinho, P. M. (2004). A genetic algorithm based approach to the mixed-model assembly line balancing problem of type II. *Computers & Industrial Engineering, 47*(4), 391-407.

Simaria, A. S., & Vilarinho, P. M. (2009). 2-ANTBAL: An ant colony optimisation algorithm for balancing two-sided assembly lines. *Computers & Industrial Engineering, 56*(2), 489-506.

Sivasankara, P., & Shahabudeen, P. M. (2013). Genetic Algorithm for Concurrent Balancing of Mixed-Model Assembly Lines with Original Task Times of Models. *Intelligent Information Management, 5*(3), 84-92.

Sparling, D., & Miltenburg, J. (1998). The mixed-model U-line balancing problem. *International Journal of Production Research, 36*(2), 485–501.

Tamura, T., Okumura, T., Singh Dhakar, T., & Ohno, K. (2011). Optimal production sequencing problem to minimise line stoppage time in a mixed-model assembly line. *International Journal of Production Research, 49*(14), 4299-4315.

Tasan, S. O., & Tunali, S. (2008). A review of the current applications of genetic algorithms in assembly line balancing. *Journal of intelligent manufacturing, 19*(1), 49-69.

Tonelli, F., Paolucci, M., Anghinolfi, D., & Taticchi, P. (2013). Production planning of mixed-model assembly lines: a heuristic mixed integer programming based approach. *Production Planning & Control, 24*(1), 110-127.

Tsai, L. H. (1995). Mixed-model sequencing to minimize utility work and the risk of conveyor stoppage. *Management Science*, 41(3), 485-495.

Tsujimura, Y., Gen, M., & Kubota, E. (1995). Solving fuzzy assembly-line balancing problem with genetic algorithms. *Computers and Industrial Engineering, 29*(1-4), 543–547.

Van Zante-de Fokkert, J. I., & de Kok, T. G. (1997). The mixed and multi model line balancing problem: a comparison. *European Journal of Operational Research, 100*(3), 399-412.

Vilà, M., & Pereira, J. (2014). A branch-and-bound algorithm for assembly line worker assignment and balancing problems. *Computers & Operations Research, 44*, 105-114.

Vilarinho, P. M., & Simaria, A. S. (2002). A two-stage heuristic method for balancing mixed-model assembly lines with parallel workstations. *International Journal of Production Research, 40*(6), 1405-1420.

Vilarinho, P. M., & Simaria, A. S. (2006). ANTBAL: an ant colony optimization algorithm for balancing mixed-model assembly lines with parallel workstations. *International journal of production research, 44*(2), 291-303.

Wilhelm, W. E., & Gadidov, R. (2004). A branch-and-cut approach for a generic multiple-product, assembly-system design problem. *INFORMS Journal on Computing, 16*(1), 39-55.

Yano, C. A., & Rachamadugu, R. (1991). Sequencing to minimize work overload in assembly lines with product options. *Management Science*,37(5), 572-586.

APPENDIX

Mixed Model Line Balancing Integer Program (AMPL)

```
# SETS
set task;
param nbrstations;
param nbrworkplaces;
set station=1..nbrstations;
set workplace=1..nbrworkplaces;
set worker={station,workplace};
set prec within {task,task};
set ss within {task,task};
set sw within {task,task};
set ns within {task,task};
set nw within {task,task};
set tasklimit within task;


# PARAMETERS
param c >= 0;                    # Cycle time
param LB >= 0;                   # Lower Bound
param t {h in task} >= 0;        # t[h] is the expected duration of task h
param q {h in task} >= 0;        # q[h] is the mounting position of task h
param es {h in task} >= 0;       # es[h] is the earliest time to start task h
param lf {h in task} >= 0;       # lfn[h] is the latest time to finish task h
param EST {h in task} = ceil(es[h]/c);
param LFN {h in task} = ceil(lf[h]/c);
param ER {h in task} >= 0;       # ER[h] is the expected ergonomic risk of
task h
param Eli1 {h in tasklimit} >= 0;
param Eli2 {h in tasklimit} >= 0;
param Eli3 {h in tasklimit} >= 0;

param Erg >= 0 ;                 # Ergonomic risk target
param ST {i in station};         # Earliest starting time of worker (i,j)


# VARIABLES

var X {(i,j) in worker, h in task} binary;
var Y {(i,j) in worker} binary;
var V {h in task, l in task} binary;
var S {h in task} >= 0 integer;

# OBJECTIVE

minimize number_of_workers:
sum {(i,j) in worker} Y[i,j];

# CONSTRAINTS
```

```
subject to C_1 {h in task}: sum {(i,j) in worker: i>=EST[h] and i<=LFN[h]}
X[i,j,h] = 1;
subject to C_3 {(i,j) in worker}: sum {h in task:i>=EST[h] and i<=LFN[h]}
X[i,j,h]*t[h] <= c*Y[i,j];
subject to C_4 {(h,l) in prec}: S[h]+t[h]<=S[l];
subject to C_5 {(i,j) in worker}: sum {h in task} X[i,j,h]*ER[h] <= Erg;
subject to C_6 {h in task}: S[h] >= sum {i in station,j in workplace:i>=EST[h]
and i<=LFN[h]} ST[i]*X[i,j,h];
subject to C_7 {h in task}: S[h]+t[h]<= sum {i in station,j in
workplace:i>=EST[h] and i<=LFN[h]} (ST[i]+c)*X[i,j,h];
subject to C_8 {h in task,l in task, (i,j) in worker: h<>l and
i>=max(EST[h],EST[l]) and i<=min(LFN[h],LFN[l])}: V[h,l]+V[l,h] >=
X[i,j,h]+X[i,j,l]-1;
subject to C_9 {i in station,h in task,l in task: (q[h]<>0 && q[l]<>0) and
q[h]==q[l] and h<>l and l>h and i>=max(EST[h],EST[l]) and
i<=min(LFN[h],LFN[l])}:  V[h,l]+V[l,h] >= sum {j in workplace}
(X[i,j,h]+X[i,j,l])-1;
subject to C_10 {h in task,l in task: h<>l}: S[h]+t[h] <= S[l]+(1-
V[h,l])*(lf[h]-es[l]);
subject to C_11 {i in station,(h,l) in ss: i>=max(EST[h],EST[l]) and
i<=min(LFN[h],LFN[l])}: sum {j in workplace} X[i,j,h]= sum {j in
workplace}X[i,j,l];
subject to C_12 {(i,j) in worker,(h,l) in sw: i>=max(EST[h],EST[l]) and
i<=min(LFN[h],LFN[l])}: X[i,j,h]=X[i,j,l];
subject to C_13 {i in station,(h,l) in ns: i>=max(EST[h],EST[l]) and
i<=min(LFN[h],LFN[l])}: sum {j in workplace} (X[i,j,h]+X[i,j,l]) <= 1;
subject to C_14 {(i,j) in worker,(h,l) in nw: i>=max(EST[h],EST[l]) and
i<=min(LFN[h],LFN[l])}: X[i,j,h]+X[i,j,l] <= 1;
subject to C_15 {(i,j) in worker:j<=card(workplace)-1}: Y[i,j]>=Y[i,j+1];
subject to C_16 {(i,j) in worker, h in tasklimit:i<>Eli1[h] and i<>Eli2[h] and
i<>Eli3[h]}: X[i,j,h] = 0;
```

Example data file:

```
set task := 1 2 3 4 5 8 9 10 15 16 17 18 19 21 23 27 29 30 31 32 33 34 35 38
41 42 44 48 55 56 58 59 60 62 64 65 67 68 69 71 73 74 75 78 79 80 83 85 86 87;
set prec := (1,3) (1,4) (1,5) (1,8) (1,9) (9,10) (15,16) (15,17) (15,18)
(15,19) (15,21) (21,23) (15,27) (15,29) (1,30) (33,31) (38,32) (1,33) (15,34)
(15,35) (15,38) (1,41) (1,42) (30,44) (29,48) (3,55) (62,58) (62,59) (60,62)
(17,64) (71,67) (71,68) (69,71) (4,73) (30,74) (1,75) (83,78) (83,79) (80,83)
(18,85) (29,86) (15,87) (16,27) (59,58) (68,67) (56,73) (79,78) (65,85);
set ss :=;
set sw := (58,59) (59,60) (60,62) (67,68) (68,69) (69,71) (78,79) (79,80)
(80,83);
set ns :=;
set nw :=;
set tasklimit := 8 9 19 21 55 64 73 85;

param nbrstations :=5;
param nbrworkplaces:=5;
param c := 103000;
param Erg := 500;
param: ST :=
1      0
2      103000
3      206000
4      309000
5      412000
;

param: t      ER     q      es ls:=
1      60     20     1      1      5
2      3300   17.90909091  1      1      5
3      13500  19.8   1      1      5
4      11340  19.57142857  3      1      5
5      4020   23.59701493  1      1      5
8      24840  22.60869565  3      1      5
9      18900  35.94285714  3      1      5
10     7020   44     1      1      5
15     60     20     9      1      5
16     3120   26.94230769  7      1      5
17     13500  19.8   7      1      5
18     11340  19.57142857  9      1      5
19     23940  22.09022556  9      1      5
21     22680  35.34920635  9      1      5
23     10620  44     7      1      5
27     3300   17.90909091  9      1      5
29     9828   18.82  9      1      5
30     10008  19.03934426  3      1      5
31     6216   19.40135135  1      1      5
32     6216   19.40135135  7      1      5
33     4536   10.15  1      1      5
34     5724   6.622641509  7      1      5
35     11592  16.38913043  7      1      5
```

```
38      5166   9.851219512  7      1      5
41      6264   6.822413793  1      1      5
42      12852  15.60588235  1      1      5
44      10122  16.18423237  3      1      5
48      10626  14.81897233  9      1      5
55      70320  24.14846416  1      1      5
56      1440   24     1      1      5
58      3960   24     0      1      5
59      4320   46     0      1      5
60      5220   12     0      1      5
62      11520  49     0      1      5
64      70320  24.12798635  7      1      5
65      1440   24     7      1      5
67      3960   24     0      1      5
68      4320   46     0      1      5
69      5220   12     0      1      5
71      11520  49     0      1      5
73      67260  26.64942016  3      1      5
74      4200   15.64285714  3      1      5
75      13500  19.04  3      1      5
78      3960   27     0      1      5
79      4320   23     0      1      5
80      5280   12     0      1      5
83      11880  90     0      1      5
85      67260  26.87421945  9      1      5
86      4200   14.91428571  9      1      5
87      13500  19.04  9      1      5
;


param: Eli1 Eli2 Eli3 :=
8       1      0      0
9       1      0      0
19      1      0      0
21      1      0      0
55      3      0      0
64      3      0      0
73      4      0      0
85      4      0      0
;
```

Mixed Model Line Balancing Constraint Program (CPLEX Optimization Studio)

```
using CP;

// Number of WorkStations
int nbWorkstations= ...;
range Workstations= 1..nbWorkstations;

// Tasks names
{int} TaskN=...;

// Tasks assignment limitation
{int} TaskLimit=...;


// Number of Stations
int nbStations=...;
range Stations= 1..nbStations;


// Cycle time
int Cycle= ...;

// Station start and end times for each Worker
int SStart[Stations]=...;
int SEnd[Stations]=...;

// Earliest time and latest finish for each task
int est[TaskN]=...;
int lfn[TaskN]=...;

// Duration, mounting position, and expected ergonomic rating for each task
int duration[TaskN]= ...;
int mount[TaskN]= ...;
float ER[TaskN]= ...;

// Target total expected ergonomic rating for each worker
int Erg= ...;

// Task assignment eligibility list
int Eli1[TaskLimit]=...;
int Eli2[TaskLimit]=...;
int Eli3[TaskLimit]=...;

// Precedence relation
tuple Precedence {
  int pre;
  int post;
};
{Precedence} precedences = ...;

// Assignment constraints
tuple AssignConst {
```

```
 int task1;
 int task2;
}

{AssignConst} ss = ...;
{AssignConst} sw = ...;
{AssignConst} ns = ...;
{AssignConst} nw = ...;

// Feasible time window
{int} bounds[i in TaskN][j in Stations] = asSet(est[i]..lfn[i]) inter
asSet(SStart[j]..SEnd[j]);

// Decision variables
dvar int usage in 0..nbStations*nbWorkstations;
dvar interval Task[i in TaskN] in est[i]..lfn[i] size duration[i];
dvar interval TaskStaWork[i in TaskN][j in Stations][k in Workstations]
optional in (card(bounds[i][j])==0 ? 0 :
first(bounds[i][j]))..(card(bounds[i][j])==0 ? 0 : last(bounds[i][j]));;
dvar sequence Seq[j in Stations][k in Workstations] in all(i in TaskN)
TaskStaWork[i][j][k];

// Objective
minimize usage;

// Constraints
subject to
{
      forall(i in TaskN) {
               alternative(Task[i], all(j in Stations,k in Workstations)
TaskStaWork[i][j][k]);
  }
// No overlap on Workers
    forall(j in Stations)
      forall(k in Workstations)
             noOverlap(Seq[j][k]);

// Same mounting position
      forall(m in 1..9)
             noOverlap( all(i in TaskN: mount[i] == m) Task[i] );

// Precednece relation
      forall(i in precedences )
             endBeforeStart(Task[i.pre], Task[i.post]);

// Same station constraint
      forall(t in ss)
        forall (j in Stations)
             sum(k in Workstations) presenceOf(TaskStaWork[t.task1][j][k]) ==
sum(k in Workstations) presenceOf(TaskStaWork[t.task2][j][k]) ;
// Not same station constraint
      forall(t in ns)
```

```
        forall (j in Stations)
             sum(k in Workstations) (presenceOf(TaskStaWork[t.task1][j][k])+
presenceOf(TaskStaWork[t.task2][j][k]))<=1;
// Same worker constraint
      forall(t in sw)
        forall(j in Stations)
          forall(k in Workstations)
                   presenceOf(TaskStaWork[t.task1][j][k]) ==
presenceOf(TaskStaWork[t.task2][j][k]);
// Not same worker constraint
      forall(t in nw)
        forall(j in Stations)
          forall(k in Workstations)

      presenceOf(TaskStaWork[t.task1][j][k])+presenceOf(TaskStaWork[t.task2][j
][k])<=1;

// Ergonomics
      forall(j in Stations)
        forall(k in Workstations)
          sum(i in TaskN) presenceOf(TaskStaWork[i][j][k])*ER[i]<= Erg;

// Eligibility constraint
      forall(i in TaskLimit)
        forall(j in Stations: j!=Eli1[i] && j!=Eli2[i] && j!=Eli3[i])
          forall(k in Workstations)
        presenceOf(TaskStaWork[i][j][k])==0;

// Breaking symmetry
      forall(j in Stations, k in 1..nbWorkstations-1)
         max(i in TaskN) presenceOf(TaskStaWork[i][j][k+1]) <= max(i in TaskN)
presenceOf(TaskStaWork[i][j][k]);

// The number of Workers used
      usage == sum(j in Stations,k in Workstations) max(i in TaskN)
presenceOf(TaskStaWork[i][j][k]);
}
```

Example data file:

```
nbWorkstations=5 ;
nbStations=5;
Cycle=103000;
Erg=500;
TaskN=
{1,2,3,4,5,8,9,10,15,16,17,18,19,21,23,27,29,30,31,32,33,34,35,38,41,42,44,48,
55,56,58,59,60,62,64,65,67,68,69,71,73,74,75,78,79,80,83,85,86,87};
TaskLimit={8,9,19,21,55,64,73,85};
SStart=[0,103000,206000,309000,412000];
SEnd=[103000,206000,309000,412000,515000];
duration=[60,3300,13500,11340,4020,24840,18900,7020,60,3120,13500,11340,23940,
22680,10620,3300,9828,10008,6216,6216,4536,5724,11592,5166,6264,12852,10122,10
626,70320,1440,3960,4320,5220,11520,70320,1440,3960,4320,5220,11520,67260,4200
,13500,3960,4320,5280,11880,67260,4200,13500];
est=[0,60,60,60,60,60,60,18960,0,60,60,60,60,60,22740,3180,60,60,4596,5226,60,
60,60,60,60,60,10068,9888,13560,0,21060,16740,0,5220,13560,0,21060,16740,0,522
0,11400,10068,60,21480,17160,0,5280,11400,9888,60];
lfn=[431180,515000,444680,447740,515000,515000,507980,515000,431180,511700,444
680,447740,515000,504380,515000,515000,504374,504878,515000,515000,508784,5150
00,515000,508784,515000,515000,515000,515000,515000,447740,515000,511040,49520
0,506720,515000,447740,515000,511040,495200,506720,515000,515000,515000,515000
,511040,494840,506720,515000,515000,515000];
mount=[1,1,1,3,1,3,3,1,9,7,7,9,9,9,7,9,9,3,1,7,1,7,7,7,1,1,3,9,1,1,0,0,0,0,7,7
,0,0,0,0,3,3,3,0,0,0,0,9,9,9];
precedences={<1,3>,<1,4>,<1,5>,<1,8>,<1,9>,<9,10>,<15,16>,<15,17>,<15,18>,<15,
19>,<15,21>,<21,23>,<15,27>,<15,29>,<1,30>,<33,31>,<38,32>,<1,33>,<15,34>,<15,
35>,<15,38>,<1,41>,<1,42>,<30,44>,<29,48>,<3,55>,<62,58>,<62,59>,<60,62>,<17,6
4>,<71,67>,<71,68>,<69,71>,<4,73>,<30,74>,<1,75>,<83,78>,<83,79>,<80,83>,<18,8
5>,<29,86>,<15,87>,<16,27>,<59,58>,<68,67>,<56,73>,<79,78>,<65,85>};
ER=[20,17.91,19.8,19.57,23.6,22.61,35.94,44,20,26.94,19.8,19.57,22.09,35.35,44
,17.91,18.82,19.04,19.4,19.4,10.15,6.62,16.39,9.85,6.82,15.61,16.18,14.82,24.1
5,24,24,46,12,49,24.13,24,24,46,12,49,26.65,15.64,19.04,27,23,12,90,26.87,14.9
1,19.04];
ss={};
ns={};
sw={<58,59>,<59,60>,<60,62>,<67,68>,<68,69>,<69,71>,<78,79>,<79,80>,<80,83>};
nw={};
Eli1=[1,1,1,1,3,3,4,4];
Eli2=[0,0,0,0,0,0,0,0];
Eli3=[0,0,0,0,0,0,0,0];
```

Mixed Model Sequencing Integer Program (AMPL)

```
# SETS
param nbrstations;
param nbrcycles;
param nbrmodels;
set station=1..nbrstations;
set cycle=1..nbrcycles;
set models=1..nbrmodels;

# PARAMETERS
param c >= 0;                                   # Cycle time
param l {k in station} >= 0;                    # Station length
param d {m in models} >=0;                      # Demand for each model
param q;                                        # Min station start
param p {m in models, k in station} >= 0;       # Processing time for car m at
station k
param e=0.1;

# VARIABLES
var X {m in models, t in cycle} binary;
var Y {k in station, t in cycle} binary;
var S {k in station, t in cycle};

# OBJECTIVE
minimize utility:
sum {k in station, t in cycle} Y[k,t];

# CONSTRAINTS
subject to C_1 {t in cycle}: sum {m in models} X[m,t]=1;
subject to C_2 {m in models}: sum {t in cycle} X[m,t]=d[m];
subject to C_3 {k in station, t in cycle}: S[k,t]+ sum {m in models}
p[m,k]*X[m,t]-          l[k]-e>=-l[k]*(1-Y[k,t]);
subject to C_4 {k in station, t in cycle}: S[k,t]+ sum {m in models}
p[m,k]*X[m,t]-l[k]<=l[k]*Y[k,t];
subject to C_5 {k in station, t in cycle:t>=2}: S[k,t] >= S[k,t-1]+ sum {m in
models} p[m,k]*X[m,t-1]-c-(l[k]*Y[k,t-1]);
subject to C_6 {k in station, t in cycle:t>=2}: S[k,t] >= S[k,t-1]-c-((1-
Y[k,t-1])*l[k]);
subject to C_7 {k in station, t in cycle:k>=2 && t>=2}: S[k,t] >= S[k-1,t-1]+
sum {m in models} p[m,k-1]*X[m,t-1]-l[k-1]-(l[k-1]-c)*Y[k-1,t-1];
subject to C_8 {k in station, t in cycle}: S[k,t] >= q;
subject to C_9 {k in station, t in cycle}: S[k,t] <= l[k]-c;
subject to C_10 {k in station}: S[k,1] = 0;
subject to C_11 {t in cycle}: S[1,t] = 0;
```

Example data file:

```
param nbrstations := 21;
param nbrcycles :=270;
param nbrmodels :=9;
param c := 175;
param q :=0;
param : l :=
1      195
2      195
3      195
4      195
5      195
6      195
7      195
8      195
9      195
10     195
11     195
12     195
13     195
14     195
15     195
16     195
17     195
18     195
19     195
20     195
21     195
;
param d:=
1      10
2      10
3      10
4      105
5      105
6      8
7      8
8      7
9      7
;
param p: 1    2      3      4      5      6      7      8      9      10     11
       12     13     14     15     16     17     18     19     20     21 :=
1      104    103    165    166    111    126    97     100    179    178    161
       96     99     147    163    163    173    176    162    164    177
2      100    103    156    175    114    121    96     97     174    172    152
       106    101    155    152    185    179    167    150    161    161
3      97     105    164    172    114    122    96     95     173    172    168
       105    102    142    156    183    178    181    152    157    154
4      92     107    161    167    115    124    93     106    178    177    167
       97     101    154    152    178    169    180    152    159    168
5      100    101    148    168    117    127    96     94     178    178    167
       101    99     146    153    169    173    172    160    162    172
```

```
6       94      108     156     167     117     130     89      102     171     177     166
        100     101     143     152     173     178     173     151     160     170
7       103     106     154     168     115     120     94      103     177     175     172
        96      96      154     154     172     174     173     155     162     167
8       109     102     164     156     111     121     101     102     171     173     157
        104     102     153     156     182     175     168     148     158     149
9       101     110     155     173     111     134     92      100     174     175     177
        96      99      155     156     171     175     184     167     157     169

;
```

Mixed Model Sequencing Heuristic Functions (MATLAB)

```
% Calculate the overload situations for a given sequence
function [a,s,f,o] = overload(nPeriods,nStations,c,l,q,x,dur)
%c = cycle time
%l = stations length (vector)
%q = station start (vector)
%x = sequence (vector)
%dur = duration
%a = total number of overload situations
%s = start times
%f = finish times
%o = overload situations
s=zeros(nPeriods,nStations);
f=zeros(nPeriods,nStations);
o=zeros(nPeriods,nStations);

 %period 1
for station= 1:nStations
    f(1,station)=s(1,station)+dur(x(1),station);
end


%station 1
for period= 2:nPeriods
    if f(period-1,1)<=l(1) && o(period-1,1)==0
        s(period,1)=max([0,f(period-1,1)-c]);
    else
        s(period,1)=max([0,s(period-1,1)-c]);
    end
    f(period,1)=s(period,1)+dur(x(period),1);
    if f(period,1)>l(1)
       o(period,1)=1;
       s(period,1)=0;
       f(period,1)=s(period,1)+dur(x(period),1);
    end

end

%complete
for station= 2:nStations
    for period= 2:nPeriods
        if f(period-1,station)<=l(station) && o(period-1,station)==0
            s(period,station)=max([q(station),f(period-1,station)-
c,f(period-1,station-1)-l(station-1)]);
        else
            s(period,station)=max([q(station),s(period-1,station)-
c,f(period-1,station-1)-l(station-1)]);
        end
        f(period,station)=s(period,station)+dur(x(period),station);

        if f(period,station)>l(station)
            o(period,station)=1;
```

```
            s(period,station)=0;
            f(period,station)=s(period,station)+dur(x(period),station);
        end
    end
end
a=sum(sum(o));

*************************************************************
*****
function [a,cost] = P_overload(nStations,c,l,q,x,dur)
%c = cycle time
%l = stations length (vector)
%q = station start (vector)
%x = sequence (vector)
%dur = duration
%a = total number of overload situations
%s = start times
%f = finish times
%o = overload situations
nPeriods=length(x);
s=zeros(nPeriods,nStations);
f=zeros(nPeriods,nStations);
o=zeros(nPeriods,nStations);

 %period 1
for station= 1:nStations
    f(1,station)=s(1,station)+dur(x(1),station);
end


%station 1
for period= 2:nPeriods
    if f(period-1,1)<=l(1) && o(period-1,1)==0
        s(period,1)=max([0,f(period-1,1)-c]);
    else
        s(period,1)=max([0,s(period-1,1)-c]);
    end
    f(period,1)=s(period,1)+dur(x(period),1);
    if f(period,1)>l(1)
       o(period,1)=1;
       s(period,1)=0;
       f(period,1)=s(period,1)+dur(x(period),1);
    end

end

%complete
for station= 2:nStations
    for period= 2:nPeriods
        if f(period-1,station)<=l(station) && o(period-1,station)==0
            s(period,station)=max([q(station),f(period-1,station)-
c,f(period-1,station-1)-l(station-1)]);
```

```
        else
            s(period,station)=max([q(station),s(period-1,station)-
c,f(period-1,station-1)-l(station-1)]);
        end
        f(period,station)=s(period,station)+dur(x(period),station);

        if f(period,station)>l(station)
            o(period,station)=1;
            s(period,station)=0;
            f(period,station)=s(period,station)+dur(x(period),station);
        end
    end
end
a=sum(sum(o));
cost=sum(max(f(nPeriods,:)-c,0));

**************************************************************
*****
% Greedy random creator
function [x]=GRC(nStations,c,l,q,dur,demand)
count=0;
avail=1:length(demand);
temp=sum(dur,2);
x=datasample(avail,1);
demand(x)=demand(x)-1;
while ~isempty(avail)
zcost=0;
over=0;
tmp=0;
for i=1:length(avail)
[over(i,1)]= P_overload(nStations,c,l,q,[x,avail(i)],dur);
end
over=[avail',over];
zcost=find(over(:,2)==min(over(:,2)));
xn=datasample(zcost,1);
x=[x,over(xn)];
demand(over(xn))=demand(over(xn))-1;
avail=find(demand~=0);
count=count+1
end

**************************************************************
*****

% Randomly swap n elements in the vector x.
function x=randswap(x,n)
l=length(x);
y = datasample(1:l,2*n,'Replace',false);
y=y(randperm(length(y)));
x(y)=x([y(n+1:2*n),y(1:n)]);
end
```

```
*************************************************************
*****
% GRASP search
function [best,Timetobest]=GRASP(nPeriods,nStations,c,l,q,dur,demand)
iter=100000;
best=0;
Timetobest=0;
xn=0;
fxn=0;
initime=cputime;
x=GRC(nStations,c,l,q,dur,demand);
f=overload(nPeriods,nStations,c,l,q,x,dur);
best=f;
Timetobest=cputime-initime;
for i=1:iter
    xn=randswap(x,1);
    fxn=overload(nPeriods,nStations,c,l,q,xn,dur);
    if fxn<=f
        f=fxn;
        x=xn;
        if f<best
           best=f;
           Timetobest=cputime-initime;
        end
    end
    fprintf('iter= %d, the current sol= %d , the best solution so far
is %d.\n',i,f,best);

*************************************************************
*****
```