MIXED-MODEL ASSEMBLY LINE DESIGN AND SEQUENCING IN A CAR BODY SHOP

ÖZGÜN AKKOL

JANUARY 2014

MIXED-MODEL ASSEMBLY LINE DESIGN AND SEQUENCING IN A CAR BODY SHOP

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÖZGÜN AKKOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

JANUARY 2014

Approval of the thesis:

MIXED-MODEL ASSEMBLY LINE DESIGN AND SEQUENCING IN A CAR BODY SHOP

submitted by **ÖZGÜN AKKOL** in partial fulfillment of the requirements for the degree of **Master of Science in Industrial Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Murat Köksalan
Head of Department, **Industrial Engineering**

Assoc. Prof. Dr. Sedef Meral
Supervisor, **Industrial Engineering Department, METU**

Examining Committee Members

Assoc. Prof. Dr. Canan Sepil
Industrial Engineering Department, METU

Assoc. Prof. Dr. Sedef Meral
Industrial Engineering Department, METU

Assoc. Prof. Dr. Ferda Can Çetinkaya
Industrial Engineering Department, Çankaya U.

Assist. Prof. Dr. Sakine Batun
Industrial Engineering Department, METU

Şakir Karakaya (M.S.)
Ministry of Science, Industry and Technology

Date:

ABSTRACT

MIXED-MODEL ASSEMBLY LINE DESIGN AND SEQUENCING IN A CAR BODY SHOP

Akkol, Özgün
M.Sc., Department of Industrial Engineering
Supervisor: Assoc. Prof. Dr. Sedef Meral

January 2014, 118 pages

Car manufacturing is usually in the form an assembly line composed of three consecutive shops: body welding and construction, painting and finally the final assembly shop. The first stage in car manufacturing that is the body shop consists of several assembly lines in parallel each of which may have several sub-lines feeding them. Assembly line design and sequencing is therefore the heart of manufacturing a car which may have several models. Our approach mainly consists of two phases: the design phase and the sequencing phase. In the design phase of the approach, we propose an integer-programming-formulation-based robust optimization model considering the mixed-model nature of the lines. The objective of the model is the minimization of the sum of the investment and variable costs of all the assembly lines in the design and operation of the car body shop only, over the life cycle of the reference car, given the forecasted annual demand of the car and its several models, the corresponding tact time, and the available types of stations making up the lines. We obtain the optimum designs of the lines via the robust optimization model using the software GAMS, which is an extension of a single model case of the same environment. We observe lower total costs for all the lines than the total costs obtained for the single-model approach.

In the second phase of our approach, we develop a genetic algorithm for the sequencing problem. The objective of mixed-model sequencing in the genetic algorithm is to have a smooth line, thus to have level utilization rates over time. The genetic algorithm provides the best fitness value for several test problems in very short computational times, when compared against the solutions obtained by the total enumeration method.

Keywords: car body manufacturing, mixed-model sequencing, robust optimization, genetic algorithms

ÖZ

OTOMOBİL GÖVDE ÜRETİM HATLARINDA KARIŞIK MODEL HAT TASARIMI VE ÇİZELGELEMESİ

Akkol, Özgün
Yüksek Lisans, Endüstri Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Sedef Meral

Ocak 2014, 118 sayfa

Otomobil üretimi, genellikle ardışık üç atölyeden oluşan montaj hattı şeklindedir: gövde kaynak ve yapımı, boyahane ve son olarak da nihai montaj hattı. Otomobil üretiminde gövde atölyesi olan birinci aşama birçok parallel montaj hattından oluşur ve herbir montaj hattını besleyen alt-montaj hatları vardır. Bundan dolayı montaj hattı tasarımı ve model sıralama, birçok modeli olan bir otomobilin üretiminin candamarıdır. Yaklaşımımız temel olarak iki evreden oluşur: tasarım evresi ve sıralama evresi. Yaklaşımın tasarım evresinde, hatların karışık-modelli doğasını dikkate alarak, tamsayılı programlama esaslı bir "dayanıklı optimizasyon" modeli önermekteyiz. Modelin amacı; hatları oluşturacak uygun istasyon tipleri, referans bir otomobil ve onun değişik modelleri için yıllık talep tahmini ile yıllık talebe karşılık gelen takt süresinin verili olduğu durumda, otomobilin sadece gövde üretim hatlarında yaşam dönemi boyunca oluşacak yatırım ve işletme maliyetlerinin toplamını minimize etmektir. Hatların optimal tasarımlarını; aynı üretim çevresinde tek-modelli durumun devamı niteliğinde olan ve GAMS yazılımı ile kodlanan dayanıklı optimizasyon modeli ile elde etmekteyiz. Tüm hatlar için, tek-modelli yaklaşıma göre daha düşük toplam maliyetli sonuçlar elde ettiğimizi gözlemlemekteyiz.

Yaklaşımımızın ikinci evresinde ise sıralama problemi için bir genetik algoritma önermekteyiz. Genetik algoritmadaki karışık-model sıralamanın amacı; zaman içinde düzgün bir hat ile çalışabilmek, ve böylece hattı oluşturan istasyonlarda eş kullanım oranlarına ulaşabilmektir. Genetik algoritma; çeşitli test problemleri için, toplam birerleme yöntemi ile elde edilen sonuçlar ile karşılaştırıldığında, en iyi uygunluk değerini çok kısa çözüm sürelerinde sağlamaktadır.

Anahtar Kelimeler: otomobil gövde üretimi, karışık-model sıralama, sağlam optimizasyon, genetic algoritmalar

To my mom…

ACKNOWLEDGEMENTS

I would like to thank my supervisor Asst. Prof. Dr. Sedef Meral for all her help, support, time and her smiling face. Without her, this thesis could have never completed. She had a solution in her sleeve whenever an impossible situation occurred. Her friendly attitude never diminished and it was a joy to sit and work with her.

I would like to thank Ignacio Grima a lot, for not saying no and helping me with the coding of the genetic algorithm with his experience in coding, in a time of need. Thanks to Deniz Erbaş for introducing me to him.

Thanks to Mustafa Savaşgan, Aras Bayram, Emre Oruç, Burak Bahçeci, Gökhan Çiftçioğlu, Fulya Erdem and Hüseyin Sever for opening their houses to me.

Thanks to Cansu Çiftarslan for believing in me.

And finally, thanks to my mom Sibel Akkol for all her support, to my father Ismail Akkol for his patience and to my sister Öykü Akkol for motivating me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Assembly line design and sequencing is the heart of manufacturing a car. Even a slight decrease in the cost of manufacturing may account for much more, given the high number of cars to be manufactured. In the light of this information, many concepts were thought of and analyzed. The major difference between today's assembly lines and the assembly lines that were first built is that nowadays lots of options are selectable by the buyer (like manual or automatic shift, sunroof or no). Considering these several options, even millions of different models of a certain car are produced. To cope with this highly diversified product portfolio without jeopardizing the benefits of an efficient flow-production, the so-called mixed-model assembly lines are utilized in car manufacturing.

Our problem of line design is more involved than a simple assembly line balancing. We address the assembly line design problem in the car body shop so as to obtain the lines that will result in the minimum total cost of manufacturing, including the investment and installation costs, over the whole life cycle of the reference car.

Setup times and therefore costs should have been reduced significantly in a mixed-model assembly line by the use of flexible machines and workers, justifying the assembly of several models of the same product in batches of one unit only. However using the mixed-model lines, the problem of model sequencing arises, since the sequence of several models of the product assembled should be determined in a planning horizon, like a shift or a day.

There are three stages of the car manufacturing process: main body construction, painting and lastly final assembly. Usually the main body production is the stage that causes most of the bottlenecks in car manufacturing. This is one of the motivations why we focus on the main body production in this study.

As the lives of products decrease and new models are introduced faster to the market, it is clear that a reconfigurable line is needed to deal with many kinds of models and product variety. Because of this, and more importantly, workers' health and safety considerations, hybrid lines consisting of both robotic and manual stations are increasingly utilized in car industry. That is why the use of robots is increasing in stages like the main body welding and construction. However, there is still work to be done in utilizing the robots in stations in an assembly line in order to be more cost effective, safer in some deleterious tasks, and more responsive to demand changes as well.

Main car body production consists of assembly areas that are connected automatically to each other. The assembly areas, usually arranged in lines, consist of stations where a number of robots and/or workers work together simultaneously on a subassembly of the body to assemble many parts/components together.

The assembly line design problem consists of the following sub-problems:

- The number of stations and their types (robotic/manual) for each subassembly line separately
- The assignment of tasks to stations

- The placement of safety spaces and buffer spaces on the whole line, considering all sub-lines and assembly lines

We focus on the first two sub-problems in the line design part of our study.

In a car body shop, there are several assembly lines feeding the main assembly line where the car body is fully erected and then transferred to the painting shop. We obtain the best line designs for the subassemblies separately in the body shop with reference to a domestic car manufacturing company. We extend the mathematical programming model that was already developed for the single-model case for the car manufacturing company, so as to take into account the mixed-model nature of the assembly lines. The mathematical model we propose is an integer programming based robust optimization model which is coded in GAMS. By means of this model, we intend to obtain a robust design for the car body shop considering all the models of a certain light commercial vehicle. We obtain line designs with less total costs than the single-model approach.

After the design problem is solved, we consider the problem of sequencing the models of the car in the car body shop. Sequencing in a mixed-model assembly line is an NP-hard problem. In this study, we attempt to solve our mixed-model sequencing problem in the car body shop using a Genetic Algorithm approach that aims to minimize the differences of utilizations of work stations over time. The proposed genetic algorithm (GA) provides the optimum sequence for most of the test problem instances the optimum sequences of which could have been obtained via a total enumeration method.

In the following chapters, the details of our study are presented, following the below outline.

Chapter 2 defines our problem and introduces our approach to the problem.

In Chapter 3, a review of literature on assembly line design is given, starting with the classification of the objectives of assembly lines. After the review of assembly line design studies, mixed-model sequencing literature is reviewed in the second part of Chapter 3; while in the third part of the chapter, we discuss the literature on robotic lines. In the fourth part of the chapter, GA literature is discussed since we solve the sequencing problem with a GA approach.

In Chapter 4, our robust optimization approach to the line design problem is presented. The environment of the problem, the integer programming model, its parameters, variables and constraints are explained in detail. Solutions obtained are discussed finally in this chapter.

In Chapter 5, we continue with the second phase of our solution approach which is the sequencing problem. Since we solve the sequencing problem with a GA, Genetic Algorithms are discussed in detail. After the review of some possible choices for the chromosome representation, initial population generation, crossover and mutation operators, replacement strategy, termination criteria, and some additional procedures like elitism, the reasons of our own choices are given. Our GA algorithm is discussed presenting its pseudo code. Then tuning of the GA parameters is explained. The test problems generated for the GA approach are defined. The performance of our GA approach is evaluated based on the results of the test problems.

Chapter 6 concludes the study by the highlights of the study and gives some suggestions for further research.

CHAPTER 2

PROBLEM DEFINITION

Our problem is in the context of manufacturing a car body in an efficient way using both robotic stations and manual stations together in a shop. Car manufacturing consists of three main stages in succession: body assembly, painting and then final assembly. However, most of the bottlenecks in car manufacturing occur in body shops rather than the paint and the final assembly stages; and bottlenecks increase the cost of manufacturing considerably, hence we focus on the body shop that is the first stage in car manufacturing.

Car body manufacturing is a flow shop type manufacturing with several assembly lines laid out in parallel feeding the final body assembly line where the car body is installed. Each assembly line corresponds to a certain subassembly of the car body. The assembly lines are fed at some stations of the line by some sub-assembly lines that produce some parts of the subassembly.

The assembly charts for all the parts and subassemblies making up the car body are known at the design stage of the car body shop. Based on the assembly charts, the precedence relationship diagrams of the parts and subassemblies' tasks are generated which are the main inputs at the design stage of the shop. The tasks are mostly spot welding operations in the body shop, while only a few tasks are of other types, like

tucker, nut assembly and pasting. The tasks which are not welding are defined in terms of the equivalent number of spot welds.
The available types of robots and other equipment needed in all tasks in the assembly of the car body are also known in advance at the design phase of the shop. Similarly, the capabilities of the robots are known; that is, a robot can perform a certain set of tasks in known standard times. There are some precedence relations among the robot types in their successive ordering on the assembly line which are also known at the design phase.

The objective in the design of the car body shop is the minimization of the discounted total cost of manufacturing at the rate of the yearly demand assumed over the expected lifetime of the car (which is a lightweight commercial car in our case). The total cost consists of the investment cost that includes the purchasing cost and installation cost of all robots and equipment used at the stations plus the cost of manufacturing with the exception of material cost.

In this study, we propose an approach for the design of a car body shop with reference to a real life case which is the manufacturing of a lightweight commercial car in a local car manufacturing company. The lightweight commercial car under consideration is manufactured in two different models in the car body shop.

In 2011 a research team at METU-IE had developed an approach for the design of the car body shop for the specific lightweight commercial car under consideration which is based on an integer programming formulation (Barutçuoğlu et al., 2011). In their formulation the mixed-model nature of the shop was ignored; and the lines in the car body shop were all designed taking into account the model with the higher work content in terms of tasks, as if the line being designed were of a single-model type. In this study we address the same problem of the car body shop design, taking into account the mixed-model nature of the shop. Hence we extend the former approach of the METU-IE research team to the mixed-model case so as to obtain a robust design for the body shop. In the robust design approach we propose, we try to minimize the difference between the cost of producing the car model on the line specifically designed for it and the cost of producing the car model on the line designed to produce all models of the car; and this difference is defined as the 'regret' function. We demonstrate our design approach for the reference lightweight commercial car and the body shop where its body is constructed.

Having obtained the car body shop design for the car under consideration, we then propose an approach for sequencing the several models of the car in the shop. But in sequencing the models of the car in the body shop, we do not consider the other following stage of car manufacturing, which is painting. Hence, the sequencing problem we address in this study is not specific to car body manufacturing; it is a generic mixed-model sequencing problem. Therefore the approach we develop can be adopted in any mixed-model assembly environment with the objective of minimizing the variations in stations' utilizations over time.

CHAPTER 3

LITERATURE REVIEW

3.1 Assembly Line Design and Balancing

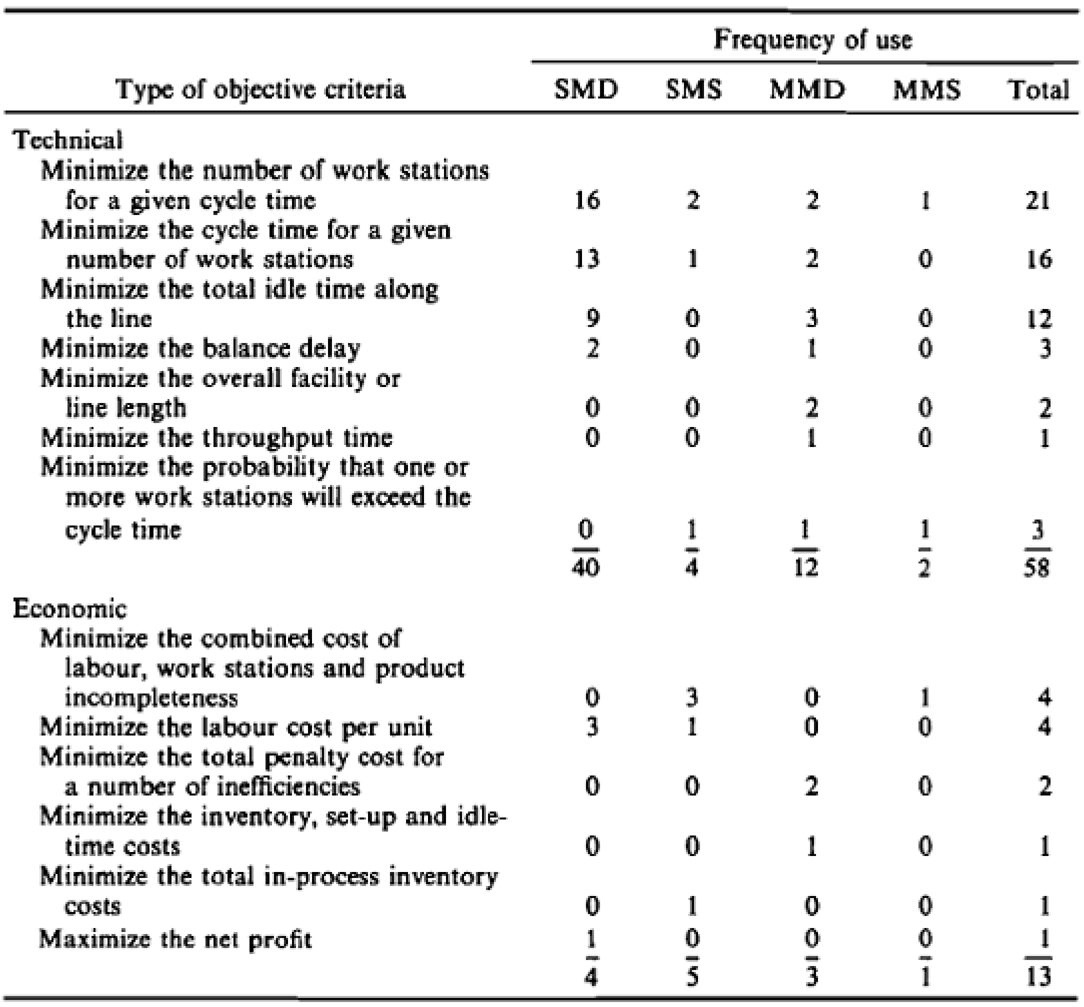
Ghosh and Gagnon (1989), in their review paper, present the table below (Table 1) for the classification and objectives of the assembly line problems, as studied in the literature.

SMD and SMS are single model lines with deterministic and stochastic task times, respectively; while MMD and MMS are mixed model lines with deterministic and stochastic task times, respectively.

For mixed model deterministic (MMD) problems, mostly priority ranking and assignment methods are used. The heuristic techniques of Nottingham University Line Sequencing Program (NULISP), Computer Method for Sequencing Operations for Assembly Lines (COMSOAL) and Computer Aided Line Balancing (CALB), address many of the factors of any ALB software available and, in all practicality, remain the most versatile, computationally reasonable (and user-friendly) techniques available for practical use. Although numerous improvements in optimization techniques and computational capabilities have been made over the 30 years of ALB research and still continue, heuristic-based programs such as COMSOAL, CALB, Mixed Model Assembly Line Balancing (MALB), NULISP and Multiple Solutions

Technique (MUST) appear to offer the only computationally efficient and versatile means for addressing real-world ALB situations.

Table 1 Objective criteria for the assembly line problems



Type of objective criteria	Frequency of use				
	SMD	SMS	MMD	MMS	Total
Technical					
Minimize the number of work stations for a given cycle time	16	2	2	1	21
Minimize the cycle time for a given number of work stations	13	1	2	0	16
Minimize the total idle time along the line	9	0	3	0	12
Minimize the balance delay	2	0	1	0	3
Minimize the overall facility or line length	0	0	2	0	2
Minimize the throughput time	0	0	1	0	1
Minimize the probability that one or more work stations will exceed the cycle time	0	1	1	1	3
	40	4	12	2	58
Economic					
Minimize the combined cost of labour, work stations and product incompleteness	0	3	0	1	4
Minimize the labour cost per unit	3	1	0	0	4
Minimize the total penalty cost for a number of inefficiencies	0	0	2	0	2
Minimize the inventory, set-up and idle-time costs	0	0	1	0	1
Minimize the total in-process inventory costs	0	1	0	0	1
Maximize the net profit	1	0	0	0	1
	4	5	3	1	13

We use an integer programming approach to design the assembly line since our conditions warrant a deterministic case and our objective is cost minimization given the tact time or the cycle time. Our problem is modeled using an economic objective criterion rather than a technical criterion. However, there are fewer papers using the economic objective criteria, although cost seems to be the most important criterion to address most of the time.

Since we deal with mixed-model situations rather than single model situations, we look for the relevant articles on this topic. This approach aims at avoiding/ minimizing sequence-dependent work overload based on a detailed scheduling which explicitly takes operation times, worker movements, station borders and other operational characteristics of the line into account.

Thomopoulos (1967) aims to minimize the total inefficiency costs. If a penalty cost per minute is associated with each inefficiency at each station, it is possible to compute the total cost of inefficiencies resulting from scheduling a unit of a given model in the sequence. He models the problem of balancing and sequencing and illustrates it by using data from the automotive industry. He observes that single model line balancing techniques are adaptable to mixed model schedules. Also, sequencing can be used to increase the efficiency of mixed model assembly lines. Even though optimality is not assured, their analysis indicates that the results are close to optimum.

In their article, Gökçen and Erel (1997) consider a goal programming approach with different goals and different priority levels which makes the proposed model have a considerable amount of flexibility for the decision maker, since several conflicting goals can be simultaneously considered. They develop their model considering a single model ALB approach. Their model is the first multiple criteria decision making approach to the mixed model problem. They find that the solution time is highly sensitive to the `number of stations` goal.

Bukchin (1998), in his article, examines which measures work better for throughput of a mixed model assembly line in a JIT environment: `Smoothed station’ measure, `Minimum idle time’ measure, `Station CV’ measure, `Bottleneck’ measure or `Model variability’ measure. In `smoothed station` measure, the objective of the balancing procedure is to minimize the fluctuation of assembly times required by each model at different stations during the shift. In `minimum idle time` measure, the objective of the balancing procedure is to decrease total idle time during a shift. In `station CV` measure, the objective function is to minimize the sum of each station’s coefficient of variation, as a performance measure for throughput. `Bottleneck` measure deals with an approximation of line cycle time for a CONWIP (CONstant WIP) controlled production system. The approximation is based on the distribution of the maximum assembly time over all stations. The `model variability measure` is a weighted sum of the variabilities of models. This study is needed, because an objective is not always clear and it is important to know the validity of the performance criterion used. As a result, the `bottleneck’ measure is found to perform better than the other measures in 12 of the 15 cases. Since each case represents a combination of line configuration and operating environment, the conclusion is that the relative quality of the ‘bottleneck’ measure is fairly robust to changes in these factors. ‘Model variability’ and ‘smoothed station’ yield the next best results. He also finds out that the absolute quality of all the measures examined decreases with an increase of line length.

Amen (1998) makes a survey of heuristic methods for cost-oriented assembly line balancing. He also presents a new priority rule called *‘best change of idle costs’*. This priority rule differs from the existent priority rules, because it is the only one which considers that production cost is the result of both production time and cost rates. Furthermore a new sophisticated method called *‘exact solution of sliding problem windows’* is presented.

Amen (2000) proposes an exact method for cost-oriented assembly line balancing. He shows that by loading the stations maximally the cost-oriented optimum can be missed. Using a variety of dominance criteria an exact backtracking method is presented. The dominance criteria are the global lower bound for the costs per product unit, the last station (if the sum of the durations of the unassigned tasks is not longer than the cycle time, the considered station is the last one), the establishment of a station with no idle costs, the two stations rule (a new station cannot be established if the sum of the durations of the operations in the current and the foregoing station does not exceed the cycle time), the potential dominance (a new station cannot be established if there exists an assignable task with a cost rate not higher than the calculated cost rate of the station under current consideration), the local lower bound for the costs per product unit and costs of all assigned tasks (a new station cannot be established if the costs per product unit of the current partial solution (i.e. the set of all assigned tasks) are not lower than those of the same set of tasks allocated to stations in a different manner earlier in the enumeration process). The backtracking method with these dominance criteria works well for small and medium-sized problem instances as it finds optimal solutions within an acceptable run time.

Bradley and Blossom (2002) consider a make-to-order (MTO) assembly line using product-mix flexibility. They propose a process to increase product mix flexibility on the assembly line through which the current demands of the marketplace can be satisfied. They also compute an upper bound on the amount of additional capacity that is required to implement MTO production. They show that MTO production and quick fulfillment are feasible in an automotive assembly plant although efficiency and cost degrade only slightly.

In their article, Rekiek et al. (2002) deal with assembly line design. They consider both cases of the given cycle times (SALBP-1) and the fixed number of workstations (SALBP-2). They see that exact methods are mainly based on branch and bound (B&B) algorithms. They notice that, despite the large number of works on assembly line design and balancing, algorithms in the literature are not heavily used by industrial companies, since, despite their effectiveness and the ease of their use, they use little data and suffer from substantial loss of information, and hence solving fictitious rather than industrial problems.

Spieckermann et al. (2004) consider a sequential ordering problem (SOP) in automotive paint shops which resembles our problem, since they are also trying to sequence car models. However, there is more than one lane that they consider, so the problem differs from ours. They attempt to find a method to find the best lane to direct a new arriving car. They show that important aspects of the problem can be modelled as the well-known SOP, and a branch and bound solution approach that exploits the main problem characteristics works well.

In their article Emde et al. (2008) try different objectives to see which of the smoothing objectives work better averaged over all problem instances. There are three alternatives to integrate the line balancing and sequencing problems. Successive planning, by executing the planning in two independent steps; simultaneous planning, by going back and forth between the two stages and adjusting accordingly; and anticipation, which aims at smoothing the workload per station over all models by using the forecasts for the models. Their first conclusion is that (irrespective of the specific criterion applied) workload smoothing considerably reduces short-term work overload compared to successive planning, where ALB disregards smoothing aspects. They find out that no criterion delivers the best performance all the time. However, they can give some clear guidelines. Vertical balancing, while still effective, is consistently less well suited to lowering work overloads than horizontal balancing. Concerning distance measures, exceedance criterion seems to be just a bit better than Manhattan and Euclidian distances which in turn is considerably better than maximum divergence. Finally they find out that both increasing product variety and forecast errors enlarge short-term work overload which is quite intuitive.

Boysen et al. (2009) survey articles about mixed model sequencing, car sequencing and level scheduling which are three major planning approaches in manufacturing. Mixed model sequencing aims at avoiding/minimizing sequence-dependent work overload based on detailed scheduling. Car sequencing is used when it is hard to collect much data. Level scheduling is used to find sequences in line with the JIT-philosophy. A hierarchical classification scheme is developed, which covers all proposed problem extensions in a systematic manner. The classification provides insights in the status quo of research in each field, also allows a comparison of the different approaches with regard to the level of planning detail and the actual problem characteristics considered. They conclude that there is a need for both theoretical and empirical results concerning the relationship among the three approaches and the resulting consequences for business practice.

In their article, Rajput et al. (2010) aim to minimize the average costs of holding, ordering, setup and backordering; while their second objective is to keep the constant consumption of each part in the assembly line. They present a cost model and compare flexible assembly line (FAL) and mixed-model assembly line (MMAL). This is the first study to compare FAL and MMAL. They find that merits go to MMAL, because MMAL is not dependent on setup time and more models produce on one single line. They also observe that the best sequence pattern and its job order, depends on production and demand quantities. The best sequence yields a continued consumption of parts and minimize the overall cost. Maximum cycle time is also a factor for higher cost. Constant demand and random demand were both set in separate occasions, according to the planning horizon. It is also notified that constant demand induces for higher cost than random demand.

In their article, Gujjula et al. (2011) aim to minimize utility work which is the amount of unfinished work caused by the moving conveyor belt. They propose a heuristic that is derived from Vogel's approximation method for transportation planning. The heuristic is able to handle large and supposedly difficult problem instances. In the end they show that the proposed heuristic significantly outperforms priority rule-based methods and requires only reasonable computational effort. However, it remains an open question whether performance can be retained if the heuristic is applied to sequencing problems with different or multiple objectives or if it can be adopted to solve related sequencing problems such as level scheduling or sequencing models on a line with limited flexibility.

Since our assembly lines have both manual and robotic stations, and thus our line designs include robots as well, we review some articles regarding robotic assembly line design.

We use an integer programming approach to design the assembly line since our conditions warrant a deterministic case and our objective is cost minimization given the cycle time. Our problem is modeled using an economic objective criteria rather than a technical criterion. However, there are fewer papers using the economic objective criteria, although cost is the most important criterion to address most of the time.

Since we deal with mixed-model situations rather than single model situations, we look at the relevant articles on this topic. This approach aims at avoiding/minimizing sequence-dependent work overload based on a detailed scheduling which explicitly takes operation times, worker movements, station borders and other operational characteristics of the line into account.

3.2 Robotic Assembly Lines

Some of the previous mentioned articles also used robotic assembly lines but the followings are the ones we examined solely to elaborate more on the robotic assembly lines.

Nicosia et al. (2002) consider the problem of assigning tasks to an ordered sequence of non-identical workstations under the constraints of precedence relations and a given cycle time. The objective is to minimize the cost of the workstations. This formulation is very similar to the robotic assembly line balancing (rALB) problem. A dynamic programming algorithm is developed for the problem, where several fathoming rules are used to reduce the number of states. The authors classify instances of the problem that are polynomially solvable. They show that minimizing the number of robots does not necessarily minimize the cost. Their model gives an efficient way to meet production requirements by taking into account the trade-off between cost of robots and performance. They show that the most significant algorithmic improvements made for assembly line balancing problem in the past 30 years can be extended to deal with assembly line design problem. Even if for general precedence graphs the problem is NP-complete, they show that a DP algorithm finds an optimal solution in polynomial time when the assembly graph width is fixed.

In their article, Kim and Park (2007) develop an integer programming formulation for robotic assembly line balancing problem and a strong cutting plane algorithm to solve it. They have special constraints because of the robotic assembly line like the limited space to store the parts and the tool capacity of robots. Moreover, the procedure gives a lower bound on the optimal solution against which the quality of the current best solution can be measured.

We use a genetic algorithm approach to solve our assembly line sequencing problem, hence we review the related literature for the genetic algorithms for mixed model sequencing below.

3.3 Genetic Algorithms for Mixed Model Sequencing

In their article Akgündüz and Tunalı (2011) review current applications of genetic algorithms in mixed model assembly line sequencing. They observe that more than half of the articles surveyed deal with the mixed model assembly line sequencing problem only, assuming that the line balancing problem was already solved. The general tendency to generate the initial population is observed to be randomization. The majority of the articles (i.e. 10 out of 13) have used elitist strategies to preserve the best individuals. As crossover and mutation operators, the majority of the researchers have preferred order crossover and inversion, respectively. When used together, these two operators have been shown to work successfully for mixed model assembly line sequencing.

Norman and Bean (1994) give an example of a genetic algorithm for scheduling problems. They provide the algorithm for an environment where there are ready times, due dates, multiple non-identical machines, routing flexibility for jobs, sequence dependent setup times, tooling constraints and has a job shop or open shop structure. Their aim is to minimize a combination of tardiness and makespan. They present their c code and show all the steps of a genetic algorithm and give basic definitions. They set their parameters and decide on their crossover and mutation techniques. They also give an example on how to solve a problem.

Levitin et al. (2004) propose a genetic algorithm for robotic assembly line balancing. They are trying to allocate equal amounts of work to the stations on the line while assigning the most efficient robot type from the given set of available robots to each workstation. They introduce both a recursive and a consecutive assignment procedure. They use the fragment reordering crossover and swap mutation operator. The consecutive assignment procedure finds better solutions but runs longer. The GA developed is shown to be consistent and robust. It achieves solutions of higher quality than a Branch and Bound algorithm, and solves large and complex problems very efficiently.

Haq et al. (2004) introduce a hybrid genetic algorithm approach to mixed-model assembly line balancing. They obtain an initial solution by using the modified ranked positional weight method and include it in the initial population of the genetic algorithm. Using this approach, they aim to reduce the search space, thereby reducing the search time. They observe that the hybrid algorithm finds the solution faster than the pure GA.

In their article Yu et al. (2005) use a multi objective genetic algorithm to schedule an assembly line. They propose a multi-objective genetic algorithm (MOGA), to research the dispatching problem with two goals: leveling the part usage rates and minimizing the makespan, and solving the multi-product dispatching problem of assembly systems. They use a pareto filtering mechanism within the genetic algorithm to eliminate non-dominate chromosomes. They use order crossover to avoid meaningless solutions and inverse mutation operator (INV operator). They find that the proposed MOGA works well through an example.

In their article, Guo et al. (2006) propose a genetic algorithm based algorithm for scheduling flexible assembly lines. They use a bi-level genetic algorithm with modified crossover and mutation operators. GA-1 generates the optimal operation assignment in three different scheduling statuses of the two-order scheduling problem, where GA-2 determines the optimal starting time of each scheduling status on the basis of the operation assignment from GA-1. Their Experimental results demonstrate that the algorithm can solve the two-order scheduling problem effectively.

Kleeman and Lamont (2007) show how to solve flow-shop, job-shop, and combined scheduling problems using multi-objective evolutionary algorithm with fixed and variable length chromosomes. They introduce a new category of scheduling problems that is quite common in real world problems that is a mixture of flow-shop and job-shop which they call the multi-component scheduling problem and present examples of the problem. Then they solve one of these examples, the engine maintenance scheduling problem. Also, chromosome representations and various crossover and mutation operators are presented. A variable length chromosome is devised in an effort to reduce the search space and is observed to perform much better than the baseline when the search space is large.

Wang et al. (2008) propose a hybrid algorithm to schedule mixed-model assembly lines with cost objectives. They use a genetic algorithm based approach. The modified order crossover (*modOX*) operator and INV mutation operator are used. The computational results show that the hybrid algorithm can always converge to the final stable state within a smaller number of generations than the GA and also the solution quality is better, especially in the case of large-sized problems.

In their article, Gao et al. (2009) present a type II robotic assembly line balancing (rALB-II) problem, in which the assembly tasks have to be assigned to workstations, and each workstation needs to select one of the available robots to process the assigned tasks with the objective of minimum cycle time. An innovative genetic algorithm (GA) hybridized with local search is proposed for the problem. They use a mixed order crossover which consists of two different crossover methods: Order Crossover (OX) and Partial-Mapped Crossover (PMX). They employ an allele-based mutation system. They conclude that the hybrid algorithm works better in especially large-size problems.

In their article Hwang and Katayama (2010) aim to maximize the line efficiency and minimize the variance of workload which is minimizing the distance between average and actual workload. They integrate the procedure of balancing and sequencing in mixed-model assembly lines. They develop an amelioration structure with genetic algorithm (ASGA). They use a weight mapping crossover (WMX) and swap mutation operator. They find out that ASGA is working better than a standard GA. Moreover, they conclude that a U-shaped assembly line works better than a straight assembly line.

Deep and Mebrahtu (2011) consider new variations of order crossover for travelling salesman problem. The travelling salesman problem is the most common problem the genetic algorithms are trying to solve. Probably because of that, new ideas are first directed to the travelling salesman problem. They propose three new variations to the OX operator and find that the existing variations work worse than the new variants and encourage the use of the new variations in the future works in genetic problems.

CHAPTER 4

AN APPROACH FOR THE ROBUST DESIGN OF THE MIXED MODEL ASSEMBLY LINES

4.1 Introduction

In this study we address two problems in the manufacture of the car bodies in the automotive industry. The first problem is the design of the car body shop at the strategical level, while the second (following) problem is the sequencing of several models of the car body on the production/assembly lines at the operational level that is discussed in Chapter 5.

In this chapter, we discuss the first problem -strategical one- and propose the solution approach specifically developed based on the real life case of a local car manufacturer. The car body shop being designed and studied in the development of our solution approach is dedicated to a certain light commercial car which has two types of models at the car body level.

4.2 The Case of the Car Body Shop

Currently the car body shop of the light commercial car has already been designed. In the car body shop we refer to in our study and in many of the car body shops as well; many sub-lines each making up a different subassembly, like sides, floor, roof, mobile parts (doors, etc.) for the car body, are matched to make up the whole car body. Hence many parallel sub-lines are connected to the final body assembly line where the whole body is constructed and then sent to the paint shop.

The car body shop is therefore a flow shop but with many parallel sub-lines feeding the final body assembly line just before the paint shop. A sub-line is also a flow line with several short lines connected to some stations of the sub-line. Each line in the body shop (either an assembly line or a sub-line) is in fact an assembly area; and these assembly areas are connected to each other mostly by automatic materials handling systems or by manual handling carried out by the workers. An assembly area is an area which contains several robotic or manual cells called stations, while a station includes usually more than one worker or robot plus some tooling that work simultaneously on a part or a subassembly of the car body.

In the stations in the car body shop, several parts/components are assembled together by several welding operations (tasks) which are manual or robotic spot welding and gas metal arc welding. Spot welds are grouped based on their position on the car body and the grouped spot welds are made one by one in succession by a single welding robot. Basically there are two types of spot welds: spot welds for attaching a new part on the subassembly and densification type spot welds that are also called respots. In densification type spot welding spot welds are made on the existing geometry of the subassembly; a new part is not attached onto the existing subassembly, the aim of respots is just to make the assembly stronger in the joining of parts together. A worker or a robot completes a group or groups of spot welds or densification spot welds in one cycle. In addition to spot welding there are other tasks carried out at the stations either by the workers manually or by robots which are basically tucker, nut , screw welding and paste operations. These operations require additional tooling that is to be held at the station like some special guns.

In the design phase of the car body shop lines, the basic technological information needed and that can be provided at this phase is as follows:

- The available types of robots and equipment that can be installed at a station and their capabilities both in performing the tasks and transferring the pieces worked on to the succeeding station on the line
- The other characteristics of the available types of robots like the area they cover, their purchasing costs and variable operating costs
- The restrictions on predecessor/successor relationships among the types of robots, if any, in the location and zoning of the them on the line
- The precedence relationships among the tasks of a subassembly or a part that are not exactly known but can be obtained from the assembly charts of the new designed car
- The capabilities or availabilities of the workers in performing the tasks

Currently two different models of the light commercial car’s body are produced in the body shop in different demand ratios in accordance with the annual demand forecasts. The light commercial car under consideration is assumed to have a certain expected lifetime.

4.3 Robust Optimization Model

An integer programming model has already been developed for the design of the car body shop by a research team at METU-IE (Barutçuoğlu et al., 2001). However, this integer programming model developed by the team is for the design of a single-model car body and does not consider the mixed-model nature of the production/assembly lines at the car body shop. Their single-model integer programming model thus developed considers the model of the car body among others which has the highest work content in the design of each assembly line.

In our study, we extend their integer programming model so as to take into account several models that are produced on the lines. Hence, we call the extended model as a robust optimization model, and the line design of the mixed-model assembly/production line thus designed as a robust design.

We intend to have a cost effective shop design such that we take into account both the first-time costs like the purchasing and installation costs of the lines plus the variable and fixed operating costs over the whole lifetime of the car under consideration.

The robust optimization model that we propose in this study is an integer programming model that minimizes the present value of the total cost of designing and installing (investment cost) plus the operating cost of the production/assembly lines in the car body shop for a planning horizon as long as the expected lifetime of the car under consideration. The solution of the robust optimization model provides the lines with the robotic and manual stations selected together with the equipment required. The whole shop is therefore designed with all its lines together with the transfer robots where necessary but with the exception of the WIP and buffer spaces in the shop. In the design of the shop the tasks are meanwhile assigned to the stations as well.

Several models of the car (in our case two models only) are considered in the design of the several assembly lines in the shop. While minimizing the costs, a robust approach is selected, because a line having the least effect from a change of the production quantities of the car models is intended. To achieve this, minimization of the maximum regret is aimed at; that is, the cost of producing a given production volume of model A car in the line developed only for it is calculated and similarly the same calculation is made for model B, and then an assembly line is designed to produce both models while minimizing the difference of costs between the new line and the line developed specifically for the individual models. In the robust model, optimum number of stations and their types are selected from a set of available station types together with the optimum assignment of tasks to the stations. Since the model developed has an economic objective function that is cost minimization the line balancing in the assignment of tasks to the stations is not the main concern.

In the reference body shop, there are two different models of the car which are called ‘combi’ and ‘cargo’. The parameters which are called as strategical parameters by the company are assumed to be like the following: working hours for a day is 21 hours, production lifetime of the car is 8 years; yearly production of combi model of car is 100,000, yearly production of cargo model is 65,000 and thus the total production is 165,000 cars yearly and the worker fee is 7,5 €/hour.

We try to minimize the maximum regret in terms of total cost which is the maximum of the two differences (i) and (ii): (i) difference between the cost of producing a combi model of the car in a line designed just for itself and the cost of producing a combi model of the car in a line designed for both models, and (ii) difference between the cost of producing a cargo model of the car in a line designed just for itself and the cost of producing a cargo model of the car in a line designed for both models. Then the maximum regret value indicates this maximum of the two regret values ((i) and (ii) above). We find the cost of producing a cargo model of the car in a line designed just for it by using the single line version of the integer programming model (Barutçuoğlu et al., 2011). We find the cost of producing a cargo model of the car in annual volume of 165,000 cars over 8 years' time in a line designed just for it by using the single line version of the integer programming model. We do the same thing for combi model of cars also. Then we use these total cost values as *z_cargo** for cargo model and *z_combi** for combi model.

In the development of our robust optimization model, we make the additional assumptions based on the reference body shop.

Assumptions

- The point in time when the body shop is designed is approximately the 30th month before the lifetime of the new car begins; in other words, it corresponds to the CAD-Phase 3 of the new car design.
- The body shop has the following characteristics: manual and robotic stations mixed on the lines; asynchronous lines that are working based on the *push-and-wait* principle; mixed-model type lines.
- The parts and subassemblies to be manufactured in house are determined; so make/buy decisions have already been made.
- The parts/subassemblies and/or tasks that are to be certainly made manually or by robots are known.
- Each part/subassembly is sellable; hence each part/subassembly has to have a dedicated independent line designed for itself only.
- A task is defined and determined as the smallest rational group of operations that cannot be decomposed to smaller tasks any further.

- Task durations depend on the type of the station (either a robot or a worker) and they are deterministic.
- The first and the last stations on a line have the capabilities of taking and leaving the base subassembly that will be processed on the line.
- The number of workers or robots at a station differs with respect to the station type.
- The time loss at the stations with more than one worker or robot due to waiting is taken into account based on the number of workers/robots at the station.
- There is an unavoidable time loss at each station's cycle time depending on the number of workers/robots at the station. This time loss is called the "constant time" which includes the waiting time for the transfer of the completed part from the station, waiting time for the take-over of the next base part to the station, loading the new part onto the station, changing the gripper or tool, if the station is a robotic station.
- In manual stations, in addition to the constant time as in the robotic stations, durations for some non-value-add operations like constant times for hanging the completed part on the hanger, and part placement are deducted from the total cycle time of the station. .
- Efficiency rate is 88% and 81% for the manual and robotic stations respectively.
- Total time of a station that can be used for the tasks assigned is determined as: efficiency rate x cycle time x number of workers or robots in the station.
- Total time of a station can then be used for the value-add tasks like spot welding, and non-value-add operations like the constant times, waiting time of workers due to simultaneous work on the same part, loading new parts on the manual station and the like.

- Lifetime of the reference light commercial car is taken as 8 years.
- The installation cost for a robotic station is 35% of the investment cost.
- Number of shifts=3/day; working hours=21 hours/day; working hours for payment to the workers=22.5 hours/day; working days/year=270 days.

4.4 Model Formulation

4.4.1 Notation of the Model

Indexes

g	task; $g \in G$
i	station; $i \in I$
t	station type; $t \in T$

Sets

I	integers upto the highest numbered station for the line
T	all station types ($T=T1 \cup T2$)
T1	manual station types
T2	non-manual –robotic– station types ($T2=T3 \cup T4$)
T3	robotic station types without transfer line
T4	robotic station types with transfer line
T5	robotic stations without transfer line that require an arrangement of their constant time when more than two pieces attached
T6	robotic stations with transfer line that require an arrangement of their constant time when more than two pieces attached
G	all spot welding tasks ($G=G1 \cup G2$)
G1	all spot welding tasks except the densification spot welding tasks

G2	densification spot welding tasks
G3	all spot welding tasks except the densification spot welding tasks (G1=G3)
G4	spot welding tasks for attaching
G5	tucker welding tasks
G6	nut welding tasks
G7	gas metal arc welding tasks
G8	liquid paste tasks
G9	screw welding tasks
G20	tasks to be done in the same station

Parameters

CT	cycle time
budget	fixed investment budget
t_space	upper limit for the area of the line considered
max_wrkr	upper limit for the number of workers for the line considered
v_m	assumed efficiency rate in manual stations (88% unless otherwise stated)
v_r	assumed efficiency rate in robotic stations (81% unless otherwise stated)
max_prt	maximum number of new parts that can be attached in the station considered
prec(g,h)	pairwise precedence relationships between tasks (if task *g* precedes task *h*, it is equal to 1, otherwise 0)
k(t)	netted (corrected) number of workers working together in *t* type station after the deduction of unavoidable delays due to simultaneous working in the station, and multiplied by the percent of productivity

wrkr(t)	number of workers working together in a *t* type station
s(g,t)	processing time for task *g* in *t* type station (seconds)
cap(g,t)	capability of *t*-typed station for task *g* (if task *g* can be done by *t*-typed station, it is equal to 1, otherwise 0.)
mpy(g)	time to place a new part manually for task *g* in any station; or time to take/drop the gun in the tasks like paste or bolt pin
mpy_t(g)	time it takes to place a new part manually by the conveyor worker in a robotic station in task *g*
eq_spt(g)	equivalent number of spot welds for each task *g* with added value (used for calculating variable costs for tasks and line automation level)
space(t)	area required for the *t* type station (m^2)
ct(t)	constant time like the hanging time for the *t* type station (seconds)
cht	constant hanging time for the first manual station in the line under consideration (seconds)
ctt	constant hanging time increasing for every manual station in the line under consideration (seconds)
adj(u,t)	matching matrix for the two stations that are next to each other in the line (if type *t* can come right after type *g*, it takes the value of 1, otherwise 0)

Cost Parameters

F1(t)	the present value of the total fixed cost of a *t* type station calculated over 8 years (=fixed operating cost + fixed investment cost)
F2(t)	the fixed investment cost of type *t* station consisting of purchasing and installation costs

v_co(g,t)	the present value of total variable operating (energy + welding electrode) cost of combi type cars calculated over 8 years when task *g* is done in a *t* type station
v_ca(g,t)	the present value of total variable operating (energy + welding electrode) cost of cargo type cars calculated over 8 years when task *g* is done in a *t* type station
c_space(t)	the cost of the area that a type *t* station covers (measured as opportunity cost)
c_bg	the cost of purchasing a bolt gun
c_tckrg	the cost of purchasing a tucker gun
c_pg	the cost of purchasing a pin gun
c_aweld	the cost of purchasing a gas metal arc welding equipment
c_pstg	the cost of purchasing a paste gun
c_c_wrkr	the present value of the total wages of a conveyor worker over 8 years
z_combi*	the total cost of producing combi model cars in the line designed only for combi model cars
z_cargo*	the total cost of producing cargo model cars in the line designed only for cargo model cars

Decision Variables - 0-1 variables

x_{git}	if a task *g* is done in *t* type *i*th station, takes a value of *1*, otherwise takes a value of *0* (defined for all tasks other than densification spots tasks)
y_{it}	if the type of the *i*th station is *t*, takes the value of *1*, otherwise takes the value of *0*

z_{gi}	if some or all of the task *g* consisting of densification spots is done in station *i*, takes the value of *1*, otherwise takes the value of 0 (defined for only densification spots tasks)
$tckrg_i$	if a tucker gun is needed in station *i*, takes the value of *1*, otherwise takes the value of *0*
pg_i	if a pin gun is needed in station *i*, takes the value of *1*, otherwise takes the value of *0*
$pstg_i$	if a paste gun is needed in station *i*, takes the value of *1*, otherwise takes the value of *0*
bg_i	if a bolt gun is needed in station *i*, takes the value of *1*, otherwise takes the value of *0*
$aweld_i$	if gas metal arc welding equipment is needed in station *i*, takes the value of *1*, otherwise takes the value of *0*
$active1_{it}$	if there is a part being attached in a *t* type *i*th station, takes the value of *1*, otherwise takes the value of *0* (0-1 variable helping in modeling)
$active2_{it}$	if there are 3 or 4 parts being attached in a *t* type *i*th station takes the value of *1*, otherwise takes the value of *0* (0-1 variable helping in modeling)

Decision variables – continuous variables

w_{git}	the ratio of task *g* consisting of densification spots, done in *t* type *i*th station
$w2_{git}$	the number of finished spots of task *g* consisting of densification spots, done in *t* type *i*th station
$occup_i$	the occupancy of station *i*: the total processing times of all tasks done in the station plus the times of the station used up without any added value (seconds)

$availt_i$	the available time in station *i* after subtracting the productivity loss (seconds)
$occup\_p_i$	the occupancy percentage in station *i*: 100*(occupancy rate / available time in station)
autom_n	the equivalent number of spots done by robots in the considered assembly
autom_d	the equivalent number of spots for all tasks in the considered assembly
autom_p	the automation percentage for the considered assembly
c_wrkr_i	the number of conveyor-workers needed in station *i* for loading the components assembled in station *i*
fxd_invst	the total fixed investment cost in the solution (for the line under consideration)
fxd_oprt	the present value of fixed operating costs occurring in the 8 years life cycle
var_oprt	the present value of variable operating costs occurring in the 8 years life cycle
z_cargo'	the present value of the total cost (investment + operating) in the 8 years life cycle of the cargo model for the considered robust line designed, using a 2% real interest rate (€)
z_combi'	the present value of the total cost (investment + operating) in the 8 years life cycle of the combi model for the considered robust line designed, using a 2% real interest rate (€)
z	the maximum regret between the total costs of a car model in the solution and in the line designed only for itself
operating_ca	the present value of total fixed and variable operating costs occurring in the 8 years life cycle of cargo model

operating_co	the present value of total fixed and variable operating costs occurring in the 8 years life cycle of combi model
space_c	the cost of total space used in the solution
used_space	the total space used in the solution

4.4.2 Integer Programming Model

$$\min z$$

$$s.to:$$

$$z \geq z\_combi' - z\_combi^* \qquad (0)$$

$$z \geq z\_cargo' - z\_cargo^* \qquad (1)$$

$$\begin{aligned} z\_combi' = & \sum_{i \in I} \sum_{t \in T} \left[F1(t) + c\_space(t) \right] y_{it} + \sum_{g \in G1} \sum_{i \in I} \sum_{t \in T} v\_co(g,t)\ x_{git} + \\ & \sum_{g \in G2} \sum_{i \in I} \sum_{t \in T} v\_co(g,t)\ w_{git} + \sum_{i \in I} (c\_c\_wrkr) c\_wrkr_i + \\ & \sum_{i \in I} \begin{bmatrix} (c\_pstg)\ pstg_i + (c\_pg)\ pg_i + (c\_bg)\ bg_i + (c\_aweld)\ aweld_i \\ + (c\_tckrg)\ tckrg_i \end{bmatrix} \end{aligned} \qquad (2)$$

$$\begin{aligned} z\_cargo' = & \sum_{i \in I} \sum_{t \in T} \left[F1(t) + c\_space(t) \right] y_{it} + \sum_{g \in G1} \sum_{i \in I} \sum_{t \in T} v\_ca(g,t)\ x_{git} + \\ & \sum_{g \in G2} \sum_{i \in I} \sum_{t \in T} v\_ca(g,t)\ w_{git} + \sum_{i \in I} (c\_c\_wrkr) c\_wrkr_i + \\ & \sum_{i \in I} \begin{bmatrix} (c\_pstg)\ pstg_i + (c\_pg)\ pg_i + (c\_bg)\ bg_i + (c\_aweld)\ aweld_i \\ + (c\_tckrg)\ tckrg_i \end{bmatrix} \end{aligned} \qquad (3)$$

$$\sum_{i \in I} \sum_{\substack{t \in T \ni \\ cap(g,t) > 0}} x_{git} = 1 \qquad \forall g \in G1 \qquad (4)$$

$$\sum_{i \in I} \sum_{\substack{t \in T \ni \\ cap(g,t)>0}} w_{git} = 1 \qquad \forall g \in G2 \qquad (5)$$

$$\sum_{g \in G2} \sum_{i \in I} \sum_{t \in T} \mathrm{x}_{\mathrm{git}} = 0 \qquad (6)$$

$$\sum_{\mathrm{i} \in \mathrm{I}} \sum_{\substack{\mathrm{t} \in \mathrm{T} \ni \\ \mathrm{cap(g,t)=0}}} \mathrm{x}_{\mathrm{git}} = 0 \qquad \forall \mathrm{g} \in \mathrm{G} \qquad (7)$$

$$\sum_{i \in I} \sum_{\substack{t \in T \ni \\ cap(g,t)=0}} w_{git} = 0 \qquad \forall g \in G \qquad (8)$$

$$\sum_{j \in I} \sum_{t \in T} j \;\; x_{gjt} \le \sum_{k \in I} \sum_{t \in T} k \;\; x_{hkt} \qquad \forall g \in G1, \forall h \in G3 \;\; \ni \;\; prec(\, g,h \,) = 1 \qquad (9)$$

$$\sum_{g \in G1} \sum_{t \in T2} s(g,t) \;\; x_{git} + \sum_{g \in G2} \sum_{t \in T2} s(g,t) \;\; w_{git} \le \sum_{t \in T2} y_{it} \;\; \left[k(t) \;\; CT - ct(t) \;\right] \qquad \forall i \in I \qquad (10)$$

$$\sum_{g \in G1} \sum_{t \in T1} \left[s(g,t) + mpy(g)\right] x_{git} \sum_{g \in G2} \sum_{t \in T1} s(g,t)\, w_{git} \le \sum_{t \in T1} \;\; y_{it} \left[k(t) CT - ct(t) - cht - i \;\; ctt\right] \;\; \forall i \in I \;\; (11)$$

$$\sum_{g \in G1} s(g,t) \;\; x_{git} + \sum_{g \in G2} s(g,t) \;\; w_{git} \le \;\; y_{it}\left[k(t) \;\; CT - ct(t) \;\right] - 14 * active2_{it} * 2 \qquad \forall i \in I, \;\; t = R4 \;\; (12)$$

$$\sum_{g \in G4} x_{git} \le 2 * active1_{it} + 2 * active2_{it} \qquad \forall i \in I, \forall t \in T \qquad (13)$$

$$\sum_{g \in G8} s(g,t) \;\; x_{git} \le \;\; y_{it}\left[2 * 0.85 * v\_r * \;\; CT - ct(t)\right] \qquad \forall i \in I, \;\; t = R20 \qquad (14)$$

$$\sum_{t \in T} y_{it} \le 1 \qquad \forall i \in I \qquad (15)$$

$$x_{git} \le y_{it} \qquad \forall g \in G1, \; \forall t \in T \quad \ni cap(g,t) > 0; \forall i \in I \qquad (16)$$

$$w_{git} \le y_{it} \qquad \forall g \in G2,\ \forall t \in T \quad \ni cap(g,t) > 0; \forall i \in I \qquad (17)$$

$$\sum_{t \in T} w_{git} \le z_{gi} \qquad \forall g \in G2, \forall i \in I \qquad (18)$$

$$|\,I\,|(z_{hi} - 1) + i\ \ z_{hi} \le \sum_{j \in I} \sum_{t \in T} j\ \ x_{gjt} \qquad \forall g \in G1, \forall h \in G2, \forall i \in I \ni prec(h,g) = 1 \qquad (19)$$

$$|\,I\,|(1 - z_{hi}) + i\ \ z_{hi} \ge \sum_{j \in I} \sum_{t \in T} j\ \ x_{gjt} \qquad \forall g \in G1, \forall h \in G2, \forall i \in I \ni prec(g,h) = 1 \qquad (20)$$

$$w2_{git} = w_{git} *\ eq\_spt(g) \qquad \forall g \in G2, \forall i \in I, \forall t \in T \qquad (21)$$

$$\begin{aligned} occup_i = & \sum_{g \in G1} \sum_{t \in T2} s(g,t)\ x_{git} + \sum_{g \in G2} \sum_{t \in T2} s(g,t)\, w_{git} + \sum_{t \in T2} ct(t)\ \ y_{it} + \sum_{g \in G1} \sum_{t \in T1} s(g,t)\ \ x_{git} + \\ & \sum_{g \in G2} \sum_{t \in T1} s(g,t)\, w_{git} + \sum_{t \in T1} (ct(t) + cht + i\ \ ctt)\ \ y_{it} + \sum_{g \in G1} \sum_{t \in T1} mpy(g)\ \ x_{git} + \\ & \sum_{t \in T1} [(v\_m) wrkr(t) - k(t)] CT\ \ y_{it} + \sum_{t \in T2} [(v\_r) wrkr(t) - k(t)] CT\ \ y_{it} \qquad \forall i \in I \end{aligned} \qquad (22)$$

$$availt_i = \sum_{t \in T2} [(v\_r)\ wrkr(t)\ CT]\ y_{it} + \sum_{t \in T1} [(v\_m)\ wrkr(t)\ CT]\ y_{it} \qquad \forall i \in I \qquad (23)$$

$$occup\_p_i = 100 * (occup_i\ \ / \ availt_i\ ) \qquad \forall i \in I \qquad (24)$$

$$autom\_n = \sum_{g \in G} \sum_{i \in I} \sum_{t \in T2} eq\_spt(g)\ \ (x_{git} + w_{git}\ ) \qquad (25)$$

$$autom\_d = \sum_{g \in G} \sum_{i \in I} \sum_{t \in T} eq\_spt(g)\ \ (\ x_{git} + w_{git}\ ) \qquad (26)$$

$$autom\_p = 100 * (autom\_n / autom\_d\ ) \qquad (27)$$

$$\sum_{t \in T} y_{i+1,t} \le \sum_{t \in T} y_{i\,t} \qquad \forall i \quad \ni\ i < |\,I\,| \qquad (28)$$

$$\sum_{g\in G5}\ \sum_{t\in\{T1\}-M8} x_{git} \le |G5|\, tckr_i \qquad \forall i \in I \tag{29}$$

$$\sum_{g\in G6}\ \sum_{t\in T1} x_{git} \le |G6|\ pg_i \qquad \forall i \in I \tag{30}$$

$$\sum_{g\in G7}\ \sum_{t=M1,M8,M10} x_{git} \le |G7|\ aweld_i \qquad \forall i \in I \tag{31}$$

$$\sum_{g\in G8}\ \sum_{t\in T1} x_{git} \le |G8|\ pstg_i \qquad \forall i \in I \tag{32}$$

$$\sum_{g\in G9}\ \sum_{t\in T1} x_{git} \le |G9| bg_i \qquad \forall i \in I \tag{33}$$

$$\sum_{i\in I}\ \sum_{t\in T}\ F2(t)\ \ y_{it} + \sum_{i\in I} \{(c\_bg)\ \ bg_i + (c\_pg)\ \ pg_i + (c\_tckr)\ tckr_i + (c\_aweld)\ \ aweld_i + (c\_pstg)\ \ pstg_i\ \} \le budget \tag{34}$$

$$\sum_{i\in I}\ \sum_{t\in T}\ space(t)\ \ y_{it} \le t\_space \tag{35}$$

$$used\_space = \sum_{i\in I}\ \sum_{t\in T}\ space(t)\ \ y_{it} \tag{36}$$

$$\sum_{g\in G4}\ \sum_{t\in T2}\ mpy\_t(g)\ \ x_{git} \le c\_wrkr_i\ \ CT \qquad \forall i \in I \tag{37}$$

$$fxd\_invst = \sum_{i\in I}\sum_{t\in T}\ F2(t)\ \ y_{it} + \sum_{i\in I}\ \{(c\_pg)\ \ pg_i + (c\_bg)\ bg_i + (m\_tckr)\ tckr_i + (c\_aweld)\ \ aweld_i + (c\_pstg)\ \ pstg_i\ \} \tag{38}$$

$$fxd\_oprt = \sum_{i\in I}\sum_{t\in T}\ [F1(t) - F2(t)]\ y_{it} + \sum_{i\in I}\ (c\_c\_wrkr)\ c\_wrkr_i\ \} \tag{39}$$

$$var\_oprt\_co = \sum_{g \in G1} \sum_{i \in I} \sum_{t \in T} v\_co(g,t)\ x_{git} + \sum_{g \in G2} \sum_{i \in I} \sum_{t \in T} v(g,t)\ w_{git} \tag{40}$$

$$var\_oprt\_ca = \sum_{g \in G1} \sum_{i \in I} \sum_{t \in T} v\_ca(g,t)\ x_{git} + \sum_{g \in G2} \sum_{i \in I} \sum_{t \in T} v(g,t)\ w_{git} \tag{41}$$

$$operating\_co = var\_oprt\_co + fxd\_oprt \tag{42}$$

$$operating\_ca = var\_oprt\_ca + fxd\_oprt \tag{43}$$

$$space\_c = \sum_{i \in I} \sum_{t \in T} c\_space(t)\ y_{it} \tag{44}$$

$$\sum_{g \in G4} \sum_{t \in T} x_{git} \le max\_prt \qquad \forall i \in I \tag{45}$$

$$y_{it} + \sum_{u \in T} y_{i-1,u}\,(1 - adj(u,t)) \le 1 \qquad \forall t, \forall i > I \tag{46}$$

$$\sum_{i \in I} \sum_{\substack{t \in T \ni \\ isgrn(t) > max\_isgrn}} y_{it} = 0 \tag{47}$$

$$x_{git} = x_{hit} \qquad \forall g,h \in G20, \forall t \in T, \forall i \in I \tag{48}$$

$$\sum_{i \in I} \sum_{t \in T2} y_{it} = 0 \tag{49}$$

$$
\begin{array}{lll}
y_{it} \in \{0,1\} & \forall i,t & (50) \\
x_{git} \in \{0,1\} & \forall g,i,t & \\
z_{gi} \in \{0,1\} & \forall g,i & \\
pg_i, aweld_i, pstg_i, bg_i, tckr_i \in \{0,1\} & \forall i & \\
active1_{it}, active2_{it} \in \{0,1\} & \forall i,t & \\
w_{git}, w2_{git} \geq 0 & \forall g,i,t & \\
occup\_p_i, occup_i, availt_i, c\_wrkr_i \geq 0 & \forall i & \\
autom\_n,\ autom\_d,\ autom\_p \geq 0 & & \\
fxd\_invst,\ fxd\_oprt,\ var\_oprt, transform,\ z \geq 0 & & \\
space\_c, used\_space \geq 0 & &
\end{array}
$$

In the below section, all the constraints of the model are explained:

(0) Makes the regret bigger than it is for the combi type cars.

(1) Makes the regret bigger than it is for the cargo type cars.

(2) Provides the present value calculated using a 2% real interest of the total cost (investment + operating) in the 8 years life cycle for the line considered in the solution for combi cars (€)

(3) Provides the present value calculated using a 2% real interest of the total cost (investment + operating) in the 8 years life cycle for the line considered in the solution for cargo cars (€)

(4) All the tasks except the densification spot welding tasks are assigned to only one station-type pair.

(5) Makes it possible for the densification spot welding tasks to be assigned to one or more station-type pairs, by dividing them among stations if needed.

(6) x_{git} is defined for every task, but since it is only valid for the tasks except the densification spot welding tasks, its value is zeroed for those tasks.

(7) A task is not assigned to a station-type pair if it cannot be done there.

(8) A densification spot welding task is not assigned to a station-type pair if it cannot be done there.

(9) Makes the tasks get assigned according to their precedence relationships.

(10) For the tasks that are assigned to a robotic station, their total time should not exceed the net working time of all the workers in a cycle after subtracting productivity loss and non-value-add time.

(11) For the tasks that are assigned to a manual station, their total time and loading of new parts, should not exceed the net working time of all the workers in a cycle after subtracting productivity loss and non-value-add time.

(12) There may be some "cycle time" related special constraints caused by equipment features (like number of robots > number of grippers).

(13) Makes the $aktif2_i$ variable, which is a 0-1 variable take the value 1 when 3 or 4 parts are added to the subassembly at station i.

(14) In the robotic station coded R20, since there are 3 workers but 2 paste guns, makes total available time for paste be calculated by using 2 workers.

(15) A station can only have one type.

(16) Every task apart from the densification spot welding tasks can be assigned to an open station that is of the capable type.

(17) Densification spot welding tasks can be assigned to an open station that is of the capable type.

(18) Makes the z_{gi} variable, which is a 0-1 variable, take the value 1 when a densification spot welding task is assigned to the respective station.

(19) Makes the task that comes after the densification spot welding tasks, be done in the station where the densification spot welding task is finished or the stations coming after that, according to the precedence relationships.

(20) Makes a densification spot welding task be done in the station where the task just before it is finished according to the precedence relationships.

(21) Makes a densification spot welding task be divided in two or more stations and save the value of how much of it is done.

(22) Provides the total time which is the total of assigned task times plus the non-value-add times in a station.

(23) Provides the available time of a station.

(24) Provides the occupancy percentage (used time/available time) of a station.

(25) Provides the spot-weld equivalent of all tasks done in a robotic station.

(26) Provides the spot-weld equivalent of all tasks done in a robotic or a manual station.

(27) Provides the automation percentage of the line considered.

(28) Station i+1 cannot be opened before station i is opened in a line.

(29) Makes the $tckr_i$ variable take the value 1, if a tucker task is assigned to a manual station apart from station type M8.

(30) Makes the smn_i variable take the value 1, if a nut task is assigned to the manual station.

(31) Makes the $gzlt_i$ variable take the value 1, if a gas metal arc welding task is assigned to the station types M1, M8 and M10.

(32) Makes the mcn_i variable to take the value 1, if a paste task is assigned to the manual station.

(33) Makes the vd_i variable to take the value 1, if a bolt task is assigned to the manual station.

(34) Makes the total fixed investment cost be lower than the budget.

(35) Makes the area for the line stay lower than the upper limit set in m^2s.

(36) Provides the area needed for the line in m^2s.

(37) If a new part is loaded in a robotic station, the number of conveyor workers needed is calculated by using the necessary time to load the parts.

(38) Provides the total fixed investment cost to install the considered line.

(39) Provides the present value of fixed operating costs occurring in the 8 years life cycle.

(40) Provides the present value of variable operating costs for combi type cars occurring in the 8 years life cycle.

(41) Provides the present value of variable operating costs for cargo type cars occurring in the 8 years life cycle.

(42) Provides the present value of total fixed and variable operating costs for combi type cars occurring in the 8 years life cycle.

(43) Provides the present value of total fixed and variable operating costs for cargo type cars occurring in the 8 years life cycle.

(44) Provides the cost of the area used by the line considered.

(45) Guarantees that the new parts added in a station does not exceed the maximum value set.

(46) Makes the stations next to each other in the line considered be consistent.

(47) Guarantees that the maximum number of workers allowed per station for the line considered is not exceeded.

(48) Guarantees that two tasks to be done in the same station are done in the same station.

(49) As a special extra constraint, makes it impossible to choose a robotic station in the line considered.

(50) Variables are continuous or 0-1 variables.

4.4.3 Inputs of the Model

We prepare excel sheets for the GAMS model to obtain the inputs from. First, we have the grouping of the similar station types. Table 2 is an example excerpt from the 'back floor' excel sheet where 'M#', 'R#', and 'T#' stand for manual station types, robotic station types without transfer line, and robotic station types with transfer line.

For example, in the table, column T1 includes the manual stations for the back floor line.

Table 2 Example Excel Sheet for 'Back Floor' Station Grouping

T1	**T2**	**T3**	**T4**	**T5**	**T6**
M1	R1	R1	T1	R4	T5
M2	R2	R2	T2	R7	T6
M3	R3	R3	T3	R11	T7
M4	R4	R4	T4		T8
M5	R5	R5	T5		
M6	R6	R6	T6		
M8	R7	R7	T7		
M10	R8	R8	T8		

Then, we group the similar tasks together in another excel sheet. Table 3 shows an excerpt from the 'back floor' excel sheet. For example, in the table, column G2 includes the spot welding tasks for the back floor line.

Table 3 Example Excel Sheet for 'Back Floor' Tasks

G1	**G2**	**G3**	**G4**
5Y01	5S01	5Y01	5T01
5Y02	5S02	5Y02	5T02
5Y03	5S03	5Y03	5T03
5Y04		5Y04	5T04
5Ms01		5Ms01	5T05
5T01		5T01	5T06

Table 4 is a part of the excel sheet that shows which station type can do which tasks. ‘1’indicates that the station can do the corresponding task, while ‘0’ indicates that the station cannot do the corresponding task.

Table 4 Sample Capability Excel Sheet for ‘Back Floor’

	M1	**M2**	**M3**	**M4**	**M5**	**M6**	**M8**	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**
5Y01	1	1	1	1	1	1	1	0	0	0	0	0	0
5Y02	1	1	1	1	1	1	1	0	0	0	0	0	0
5Y03	1	1	1	1	1	1	1	0	0	0	0	0	0
5Y04	1	1	1	1	1	1	1	0	0	0	0	0	0
5Ms01	1	1	1	1	0	1	0	0	0	0	0	0	0
5T01	1	1	1	1	0	1	0	0	1	0	1	1	0
5T02	1	1	1	1	0	1	0	0	1	0	1	1	0
5D01	1	1	1	1	1	1	0	1	1	1	1	1	1
5D02	1	1	1	1	1	1	0	1	1	1	1	1	1
5S01	1	1	1	1	0	1	0	1	1	1	1	1	1

Table 5 is part of the excel sheet that shows task durations.

Table 6 shows the strategic decisions like how much of a model is to be produced, and the cycle time derived from that.

Table 5 Sample Excel Sheet of Task Durations for 'Back Floor'

	M5	**M6**	**M8**	**R1**	**R2**	**R3**
5S01	476	476	476	322	322	322
5T03	50.4	50.4	50.4	31.2	31.2	31.2
5T04	50.4	50.4	50.4	31.2	31.2	31.2
5Ms02	4.638	4.638	4.638	2.68	2.68	2.68
5Ms03	4.638	4.638	4.638	2.68	2.68	2.68
5T05	25.2	25.2	25.2	15.6	15.6	15.6
5T06	25.2	25.2	25.2	15.6	15.6	15.6

Table 6 Sample Excel Sheet of Strategic Decisions for 'Back Floor'

Cycle Time (sec.)	124
Budget (€)	100,000,000
Total Space (m^2)	1,000,000
Number of Maximum Workers for a station	4
Number of Years of Production	8
Production per Year	165,000
Combi Production per Year	100,000
Cargo Production per Year	65,000

4.5 Discussion on the Results: Single Model vs. Mixed Model

We solve our model for all subassemblies/parts of the car body that the single model line is designed for. The results are then compared and discussed.

The solution sheet shows the total cost of producing the cars and the cost of producing them if all car bodies produced were combi or cargo. The solution also gives us the automation percentage which is the proportion of tasks completed by robotic stations to all the tasks done for the subassembly or part on the designed line, in terms of the number of spot welds. The number of stations are given and it is shown if they are robotic or manual. Also the number of workers are shown since more than one worker can work at a station. In Table 7, we have the first portion of the sample solution excel sheet of ‘right shroud’.

Table 7 Sample Solution excel sheet for the ‘right shroud’ by the robust optimization

max regret (€)	393,562.9
Total investment and operating cost(€)	2,229,418
Total investment and operating cost(€) for combi	1,944,433
Total investment and operating cost(€) for cargo	1,706,793
Total investment cost(€)	454,500
Operating cost combi(€)	1,445,833
Operating cost cargo(€)	1,208,193
Total Operating cost(€)	1,774,918
Automation %	51
Number of Robotic stations	1
Number of Manual stations	1
Number of stations	2
Number of robots	1
Number of workers	2
Number of workers + robots	3

In Table 8, the stations of the solution in the excel sheet can be seen. In the first line, the type of the station is given.

Table 8 Stations for the ‘right shroud’ by the robust optimization

	1st	2nd
Type of the station	M2	R18
Number of workers	2	1
Time used (sec)	212.8728	69.86
Time without added value (sec)	125.2728	18
Time of tasks (sec)	87.6	51.86
Usable time (sec)	217.728	100.2044
Station utilization (%)	97.77006	69.71752
Work done in terms of spot welds	22.5	23.47

If we search for the differences between mixed model lines and single model lines, ‘dashboard pool’ and ‘back stop riser’ are the most extreme examples in terms of the maximum regret values. In Table 9, the example is the ‘dashboard pool’. We show the combined solution sheets for the two different lines: single-model line and mixed-model line. Here we have a total cost difference of nearly 700,000 €. It is interesting to note that the lines designed are completely different: while the single model line is fully manual, the mixed model line is mostly robotic. Because of this, the investment cost of the mixed model line is 500,000 € more. But the design obtained with the mixed model approach more than makes up for this extra investment cost in the operating cost portion. In Tables 10 and 11, we list the stations of both the single model line and the mixed model line.

Table 9 Combined solution sheets for the subassembly ‘dashboard pool’

	single model line	mixed model line
max regret(€)		550,446
Total investment and operating cost(€)	5,772,341.2	5,059,433
Total investment and operating cost(€)combi		3,940,308
Total investment and operating cost(€)cargo		3,678,134
Total investment cost(€)	577,250	1,066,000
Operating cost combi(€)		2,744,333
Operating cost cargo(€)		2,482,159
Total Operating cost(€)	5,195,091	3,993,433
Automation %	0	74.8
Number of Robotic stations	0	4
Number of Manual stations	4	1
Number of stations	4	5
Number of Robots	0	4
Number of workers	6	2
Number of workers + robots	6	6

Table 10 Single model line stations for the 'dashboard pool'

Station type	M2	M2	M1	M10
Number of workers	2	2	1	1
Time used (sec)	157.6	159.7	77.9	71.1
Time without added value (sec)	102.4	98.4	47.6	23.1
Time of tasks (sec)	55.2	61.3	30.3	48.0
Usable time (sec)	159.7	159.7	79.8	79.8
Station utilization (%)	98.7	100.0	97.6	89.0
Work done in terms of bolts	13.5	15.9	12.1	12.1

Table 11 Mixed model line stations for the 'dashboard pool'

	M2	R15	R2	R35	R17
Number of workers	2	1	1	1	1
Time used (sec)	159.6	18	69.4	66.2	50.12
Time without added value (sec)	104.4	18	46	36.1	18
Time of tasks (sec)	55.2	18.0	23.4	66.2	32.12
Usable time (sec)	159.7	73.5	73.5	73.5	73.5
Station utilization (%)	99.94	24.5	94.4	92.6	68.2
Work done in terms of spot welds	13.5	3	9	8.5	20.6

On the contrary, we obtain some line designs for which the total costs obtained by the single model and the mixed model approaches are very close. In Table 12, we have such an example which is the ‘back stop riser’.

Table 12 Combined solution sheets of the subassembly ‘back stop riser’

	Mixed model line	Single model line
max regret(€)	190,524	
Total investment and operating cost(€)	1,872,860	2,067,594
Total investment and operating cost(€)combi	1,296,083	
Total investment and operating cost(€)cargo	1,861,836	
Total investment cost(€)	326,750	326,750
Operating cost combi(€)	929,732	
Operating cost cargo(€)	1,495,486	
Total operating cost(€)	1,546,110	1,740,844
Automation %	41,2	46,7
Number of Robotic stations	1	1
Number of Manual stations	1	1
Number of stations	2	2
Number of Robots	1	1
Number of workers	2	2
Number of workers + robots	3	3

We can see here that the difference between total costs by the two lines is nearly 200,000 €. In Tables 13 and 14 below, we look at the details of the solution sheet.

Table 13 Single model line stations for the 'back stop riser'

Station type	M2	R16
Number of workers	2	1
Time used (sec)	179.4	50.2
Time without added value (sec)	112.2	18
Time of tasks (sec)	67.2	32.2
Usable time (sec)	217.7	100.2
Station utilitization (%)	82.4	50.1
Work done in terms of spot welds	16	14

Table 14 Mixed model line stations for the 'back stop riser'

Station type	M2	R16
Number of workers	2	1
Time used (sec)	150.2	34.1
Time without added value (sec)	108.2	18
Time of tasks(sec)	42	16,1
Usable time (sec)	217.7	100.2
Station utilization (%)	69	34
Work done in terms of spot welds	10	7

As we observe here, the lines obtained by the two approaches (single model and mixed model) are exactly the same. The cost difference comes from the assumption of the single model line case where it is thought like all the cars produced were the model of the car with the largest work content. The differences of work done in terms of spot welds and station utilizations are exactly because of the same reason.

Table 15a, 15b and 15c are the tables showing a summary of the costs, number of stations and the automation percentage for all the lines in the car body shop obtained by the two approaches.

The solution sheet of the 'dashboard pool' of the car is given as a sample output in Appendix A.

Table 15a Summary of the results for the subassemblies: instrument sheet, back stop riser and outside panel sheet

	instrument sheet		back stop riser		outside panel sheet	
	single	mixed	single	mixed	single	mixed
Total Cost (€)	8,438,149	7,351,256	2,067,594	1,872,860	4,644,363	3,476,298
Operating Cost (€)	5,903,899	5,046,256	1,740,844	1,546,110	4,118,612	2,950,548
Investment Cost (€)	2,534,250	2,305,000	326,750	326,750	525,750	525,750
# of stations	8	9	2	2	3	3
Automation %	94	98	47	41	31	31

Table 15b Summary of the results for the subassemblies: right shroud, dashboard pool and left shroud

	right shroud		dashboard pool		left shroud	
	single	mixed	single	mixed	single	mixed
Total Cost (€)	2,268,704	2,229,418	5,772,341	5,059,433	2,122,462	2,059,838
Operating Cost (€)	1,814,204	1,774,918	5,195,091	3,993,433	1,339,962	1,722,088
Investment Cost (€)	454,500	454,500	577,250	1,066,000	782,500	937,750
# of stations	2	2	4	5	3	3
Automation %	51	51	0	75	100	100

Table 15c Summary of the results for the subassemblies: inside framework back shroud, side panel and inside right framework

	inside framework back shroud		side panel		inside right framework	
	single	mixed	single	mixed	single	mixed
Total Cost (€)	4,105,274	3,269,381	9,050,540	7,372,591	5,657,023	4,472,073
Operating Cost (€)	3,631,524	2,795,631	6,069,540	5,187,591	4,981,523	3,796,573
Investment Cost (€)	473,750	473,750	2,981,000	2,185,000	675,500	675,500
# of stations	3	3	7	6	4	4
Automation %	36	36	100	100	44	31

CHAPTER 5

A GENETIC ALGORTIHM PROPOSED FOR THE MIXED-MODEL SEQUENCING PROBLEM

In the first phase of our approach, a strategic problem is addressed, that is, car body shop is designed taking into account the mixed model nature of the lines. Now in the second phase of our approach, an operational problem is addressed. We intend to develop mixed-model sequences of the car models so as to utilize the lines in the most efficient and effective way in meeting the expected average annual demands of several models over the life cycle of the car.

Car sequencing with several models takes into account the constraints imposed by the paint shop and is known to be an NP-hard problem in the strong sense (Kis, 2004).

Actually considering the three stages in car manufacturing as car body shop, paint shop and the final assembly, car sequencing problem is a more critical problem in the final assembly stage of car manufacturing where the model variety is highest in car manufacturing. However, in car body shop where the chassis of the cars are manufactured, the model variety is at the lowest level, for example: 2-3 models of a certain car. Having reviewed the relevant literature on mixed-model sequencing problem, we propose a GA-based method to solve the general mixed-model sequencing problem that is not specific to car sequencing problem.

In this chapter, we first discuss the fundamentals of the genetic algorithm. Then we present our GA method for the mixed-model sequencing problem with all its technicalities. Finally we conclude with the results and their discussion.

5.1 Fundamentals of Genetic Algorithms

Genetic algorithms are search methods inspired from the genetic processes of the nature. They were first introduced by Holland and his colleagues at Michigan University and the basic principles were first published in Holland (1975). Genetic Algorithm (GA)-based optimization techniques are effective and have been applied widely to find the global optimum region due to its global perspective and inherent parallelism (Goldberg, 1989). They are widely used for NP-hard problems, since they are difficult to solve by the use of mathematical modeling. The fundamentals of GA can be summarized as:

- The starting points of GAs are chromosomes, which are made up of genes that have the basic information of the decision variables in the solution. The chromosomes, that are individuals of the population, then, are solutions for the problem which can be represented in various ways. These individuals, like in the nature, compete to reproduce and survive. As the evolution theory states, better individuals have more chance to pass their genes to the next generations.
- The probability of survival and reproduction in the nature is based on how good the genetic material of the individual is. In the GA, ‘how good an individual’ is calculated, since it is the objective function value of the solution the chromosome gives and is called the fitness of the chromosome.
- An initial population is generated at the beginning, having some number of individuals with different genetic characteristics. Through generations the

population evolves like the processes in the nature. There is a reproduction process that determines the genetic characteristics which would be passed on to the next generation out of all the individuals of the population. Crossover operator is used to select two parents and make a number of off-springs from them. Mutation operator is used to mutate a chromosome and add the mutated chromosome to the population. Each population updated by any number of new chromosomes is called a generation.

- The genetic process evolves without knowing the problem it is trying to solve, like it is in the nature. However, to make its job easier, more suitable crossover and mutation operators can be chosen from a number of options using problem specific information. This approach helps to get better solutions faster.

Genetic algorithms can be used to solve many kinds of problems. Since it is a general framework to work with, some key components should be decided beforehand. These may be called the fundamentals of genetic algorithms and are presented below:

- To define what the genes and the chromosome stand for
- To create an initial population
- To define the fitness function
- To identify the parent selection technique
- To find the appropriate crossover and mutation operators
- To define the replacement strategy: who the new generation would consist of after the crossover and mutation
- To decide on parameters like the population size, crossover and mutation rates

- To decide how the program ends, like the maximum number of generations specified or the specified maximum number of generations where the population fitness (the fittest chromosome in the population) does not improve any more
- If to use additional procedures like elitism or not; and if so, their size

5.2 The Proposed Genetic Algorithm

In this section, the GA terminology and how the different aspects discussed above are decided in our approach are explained.

5.2.1 Chromosome Representation

Chromosome representation is one of the most critical aspects of the GA approach. It must represent any possible solution to the problem and it should be clear from the chromosome how to calculate the fitness of the chromosome.

The important question is whether a binary or an integer representation should be used. Using a binary representation is easier, but an integer representation can contain more information and more complex information.

Different from most of the sequencing chromosomes where the jobs are ordered, in our problem, different models of the car would be ordered. An example chromosome for the ordered jobs is:

3	1	2	5	4

where the third job is done first, followed by job 1, and so on. The chromosome for our mixed-model sequencing problem with two models would look like the following:

A	B	A	A	B

where model A car comes first, followed by model B car and so on. There are two different models of the car in the example, but it can contain as many different models as required in it. Both examples have five genes, and if needed, as many as needed, can be added.

5.2.2 Initial Population Generation

The initial population is where the search for the solution starts. It is one of the first things the algorithm executes and goes on from. It is important to have a wide range of individual characteristics in the initial population in order to improve faster.

The most common method to generate an initial population is to randomly generate it. It is done easily and quite fast, even though some control schemes are invented to generate a better population, i.e., a population that has a wider range of individuals to better cover the solution space. These control routines are mostly used with non-binary chromosomes.

In our problem, basically making the population bigger can work easily if we want to minimize the chance of getting trapped in a certain region of the solution space, since our genes are straightforward.

5.2.3 Fitness Function Evaluation

The purpose of the fitness function in our GA approach is to have a sequence that minimizes the workload fluctuations of stations through time so as to have a smooth line even though the car models in the sequence are changing over time. This can also be expressed as the fluctuations in the utilizations of the stations. After reviewing the relevant literature, we decide on our fitness function as: maximization of the total of the immediate differences of utilizations of stations over time. We define the fitness function below in (1).

Notation:
n: Number of jobs
m: Number of stations
i: Station index
j,*h*: Job position index
$t_{i\,[j]}$: Processing time of the job in position [*j*] on station *i*
s_i: cycle time of station *i*
$U_{i\,[j]}$: utilization of station *i* having processed the job in position [*j*]
z: fitness function value

$$\max z = \sum_{i=1}^{m} \sum_{j=1}^{n} \left| U_{i\,[j]} - U_{i\,[j-1]} \right|$$

with

$$U_{i\,[j]} = \sum_{h=1}^{j} t_{i[h]} / j \ \ s_i \quad \forall i, j \tag{1}$$

For example, consider station 1 with three models to be processed on the line and s_1=10 seconds; models A, B, and C have processing times of 8, 4 and 6 seconds, respectively; and the sequence is A-B-C. Then $U_{1[1]}$ = 8/10; $U_{1[2]}$ = (8+4) / (10+10).

So $U_{1[2]} - U_{1[1]}$ = 1/5 (in absolute value). Then we add all the ($U_{i\,[j]}$ - $U_{i\,[j-1]}$) values for all stations over all positions on the stations and this is the fitness of the chromosome. For this example, sequence ABCABC has a fitness of 3/10 and sequence ABCCAB has a fitness of 7/25. Hence, the bigger the fitness (total of the difference of utilizations), the better the sequence is.

Based on this fitness function, if a station uses less time for a model, the next model has to be a more time consuming one to even the workload of the station.

5.2.4 Genetic Operators and Techniques

These are playing the key role in the GA as they may be the sole determinants on how the algorithm will perform.

5.2.4.1 Parent Selection

This is how the parents will be chosen from the population to make an offspring. The offspring will carry the characteristics of the parents as it is in the nature. From the early times of the GA literature until now, many different methods have been introduced, from random selection like the roulette wheel selection to more complex methods like the tournament selection. The techniques are described in three headings below:

Roulette Wheel Selection. First we calculate the fitness values of the individuals. Then parents are selected according to their fitness values. The better the fitness value of the individual, the more the chance for it to be chosen for being a parent. It works like this:

- We add all the fitnesses of the individuals together.

- We create a random number from 0 to the sum found in the first step.
- We then start to add the fitness of the chromosomes together until we reach the random number from the second step and the first chromosome after we exceed that number is our parent chromosome.

This is really an easy technique to use, but it has some drawbacks. Even though it is easy for the better individuals to shine in the early generations, when the difference between the chromosomes gets smaller, the generations do not grow well. Also, if the difference between the better chromosome and others are too big, the probability of selecting other chromosomes but the dominant chromosome to be the parent is small. However, these are not important drawbacks for our problem, since we do not have time issues and there cannot be a big difference among the fitnesses of the chromosomes.

Rank Selection. In this technique, we rank the chromosomes according to their fitness. The worst fitness gets a new fitness of 1 and the second worst 2, and this process goes on like this to the best fitness having the number that is the population size. This process can be modified, of course, to suit the user’s needs. Thus, this small interference arranges that the worst chromosomes have a bigger chance of being selected and hence not getting stuck with the best chromosomes. The drawback of this technique is that it requires more computation, and since the differences in the chromosomes are made less, it takes a longer time to converge.

Tournament selection. This technique chooses two or more individuals in the population and selects the fittest one as a parent. There are deterministic and probabilistic versions of this method. In the deterministic version, the fittest one is chosen for sure, whereas in the probabilistic version, a probability is attached to see what portion of the time the fittest one is chosen. Its computational effort is not much, since it only needs a preference in a small set of chosen individuals, so this idea is one of the new popular ideas.

Elitism. There is not so small a chance that when a new generation is entered, the best chromosome from the old generation will be lost. To prevent this problem, the idea of elitism is introduced. When it is used, a number of elite (the chromosomes with the best fitness values) individuals from the last generation are saved for the next generation. Elitism can very rapidly increase the performance of GA, because it prevents losing the best found solution.

The pros and cons of the selection techniques are explained above and when the literature is examined, the problems like ours mostly choose the roulette wheel selection. Since elitism is an easy concept to use, we decide to use the roulette wheel technique with elitism.

5.2.4.2 Crossover Operator

Employing effective crossover operators is important for combining existing solutions with new ones and for generating diversity. The former can be implemented by a good crossover operator, and the latter can be implemented by the mutation operator. The crossover operator takes the genetic material from the parents and makes one or two offsprings with it. The type of the operator is important in establishing which information will be gained from which parent and how it is put into the offspring.

The operator can be used easily to put in problem specific information. Nevertheless, the most commonly used operators are single-point crossover and two-point crossover operators which do not consider any problem characteristics.

A two-point crossover operator can be seen in the figure below:

Parent 1	**A**	**B**	**A**	**A**	**B**
Parent 2	B	A	B	A	A
Child 1	**A**	A	B	**A**	**B**
Child 2	B	**B**	**A**	A	A

Figure 1 A Two-Point Crossover Operator

As the GA literature advanced, more crossover operators were introduced. Uniform operator is one of them, where the offspring will have half the genes of the first parent and half the genes of the second parent if the mixing ratio is 0.5. Crossover points are randomly chosen. This approach seems poor, but evidence suggests that it is a more exploratory approach than the traditional approaches.

There is also the order crossover (OX) which is getting more and more common in the last years. Many of the articles we studied use this crossover method.

It starts by randomly choosing two cut points on the parent chromosomes. In order to create an offspring, the string between the two cut points in the first parent is first copied to the offspring. Then, the remaining positions are filled by considering the sequence of jobs in the second parent, starting after the second cut point (when the end of the chromosome is reached, the sequence continues at position 1). The example of a job sequencing in this chromosome type looks like this:

parent 1 : 2 7 | 1 3 4 | 5 8 6

parent 2 : 1 8 | 2 5 6 | 3 7 4

---

offspring

(step 1) : - - 1 3 4 - - -

(step 2) : 2 5 1 3 4 6 7 8

Figure 2 An order crossover

In the figure above, after copying jobs 1, 3 and 4 to the offspring, jobs 2, 5 and 6 are put into the offspring starting from the beginning. Then, jobs 3, 7 and 4 are tried, but since jobs 3 and 4 were already in the offspring, only job 7 could be entered. Lastly, job 8 was entered to the last place in the chromosome.

Since we decided to use the OX operator, we saw that it did not fit exactly with our chromosome structure. Actually none of the operators do. Thus, we had to modify it to support our needs. It starts the same way as the OX operator, but since we do not have different numbers for all the places in the chromosome, we had to decide what to do when an overlap occurs. We solved this problem by looking at the chosen part of the first parent and eliminating them in the second parent starting from the beginning. Then the other genes are put in like in the order crossover modifier. An example for sequencing of three models, A, B and C is shown below:

parent 1 : A B | A C B | A B C

parent 2 : C B | A B A | C A B

---

offspring

(step 1) : - - A C B - - -

(step 2) : B A A C B C A B

Figure 3 The Order crossover modified for our needs

5.2.4.3 Mutation

To maintain diversity, a mutation operator should be used. If a mutation operator is not used, it is highly likely that the GA converges to a local optimum. To avoid this entrapment, we have to randomly change some chromosomes, and thus maintain diversity.

The most common mutation operator in the literature from the early stages of the GA literature is the perturbation mutation operator which is a probabilistic operator. So, it randomly chooses a chromosome and one or more genes from it. Then it basically interchanges its value, 0 to 1 and 1 to 0 in a binary chromosome or any value to any other working value in a non-binary one.

An interesting alternative mutation operator is the boundary mutation operator. It changes a randomly chosen non-binary gene to the upper or lower bound (also chosen randomly).

The uniform mutation operator is just like the boundary mutation operator, but the user specifies the upper and the lower bounds she likes to work with. The nonuniform mutation operator works in the same way. The only difference is that probability of the amount of mutation is more in the early stages of the algorithm and it drops towards the end. This way, it lets the algorithm fine-tune in the late stages.

The inversion mutation is getting more popular in the last years. We choose to use the inversion mutation in our GA. It works by choosing a chromosome and two cut points. It then inverts what is between the cut points, like this:

Original chromosome:	2 7 | 1 3 4 | 5 8 6
After the mutation operator:	2 7 | 4 3 1 | 5 8 6

Figure 4 The inversion mutation

5.2.4.4 Replacement

Replacement strategy is used to determine which chromosomes will be used to make up the next generation. Since we are trying to improve the best solution in the population, there are various strategies to update the population. One important factor is the generation gap, which is the proportion of the individuals to be replaced in every generation.

There are two strategies to decide on the generation gap in the literature. The first strategy is to use the generation gap as ‘1’, which means to replace the whole population. This strategy is called the generational or non-overlapping replacement. Elitism is especially used in this strategy to eliminate the loss of fit individuals when passing on to the next generation.

The second strategy is called the steady-state or overlapping replacement. While inserting some individuals into the population, it is important to decide on the individuals of the population that will be replaced with the new ones. To decide on this, there is a number of different methods like replacing randomly chosen ones, replacing the worst ones, replacing the oldest ones, replacing the parents or choosing by the Kill Tournament. Smith and Vavak (1999) make a comparative study on these alternatives and state the benefits, deficiencies and how much computational effort is needed for each of these methods. They make exact Markov models for some and appropriate Markov models for other replacement strategies; and with the help of simulations, general conclusions for different strategies are drawn.

There is no evidence in literature about one strategy being better than another one. Since we are using elitism, we find it fit appropriate use the generational strategy. That way, we are not losing the best individuals, while letting new fit individuals in the population immediately.

5.2.4.5 Termination

In most GA applications, a stopping criterion is designed based on convergence. After calculating through a specific number of iterations, the algorithm terminates. In GA terms, we stop after a satisfactory number of generations. In our study, we have two stopping criteria which are stopping after 400 generations or stopping if the same chromosome stays fit after 50 consecutive generations. These numbers can be changed if needed, but are found to be satisfactory as they give the best fitness in every test.

5.2.5 Parameter Determination

There are important parameters that affect the performance of the algorithm. These are population size, crossover and mutation rates. They go hand in hand with each other, so we have to decide on all of them simultaneously.

The population size is the number of individuals generated and it is the same throughout all generations. The crossover and the mutation rates show how frequently a chromosome would be updated using those operators.

To decide on the population size, the first factor used is the length of the chromosomes. There are some ideas for how the length of the chromosomes should affect the population size. There are linear and exponential calculations to help decide. When the length of the chromosome gets longer, the exponential relations do not tend to work well. They result in impractically large population sizes. One of the most basic and most supported ideas is for a chromosome length of *n*, to use a population size in the range from *n* to *2n*.

The drawback of a small population is the possibility of it converging too quickly without sufficient exploration and the drawback of a large population is that the time to find the solution can get too long. To balance these, higher crossover and mutation rates are advised for small populations and lower rates are advised for large populations. Also, larger populations are better if “replace worst” or “steady state replacement” options or big percentage for the elitism option are chosen, since there may be premature convergence with these options.

5.2.5.1 The Tuning of the Mutation and Crossover Rates

Here the goal will be to find the best parameter values to get the best possible performance from the algorithm. We can safely fix the population size to 2*n* where the chromosome length is *n*. Then we try different crossover and mutation rates to see which ones perform better. Here is a table showing the fitness values for different mutation and crossover rates. Here we are running the algorithm for 800 generations or until a chromosome is the fittest for 200 generations. Table 16 shows these different trials for the crossover and mutation rates.

Table 16 Tries for crossover & mutation rates

product mix for 3 models	44-33-23	32-9-9	12-7-6	12-5-3
crossover, mutation rates	fitness	fitness	fitness	fitness
0.7, 0.01	2.09632	1.71446	1.49118	1.07667
0.8, 0.01	2.09457	1.71533	1.49078	1.07667
0.6, 0.01	2.09780	1.71199	1.49188	1.07667
0.5, 0.01	2.09863	1.71737	1.49103	1.07667
0.4, 0.01	2.09754	1.71743	1.49017	1.07667
0.55, 0.01	2.09880	1.71804	1.49196	1.07667
0.55, 0.02	2.09878	1.71783	1.49073	1.07667
0.55, 0.03	2.09657	1.71732	1.49037	1.07667
0.55, 0.05	2.09514	1.71694	1.49001	1.07667

Since we know that a bigger fitness is a better one, we can say that a crossover rate of 0.55 is working best, whatever the size of the problem is. After that, since we know the best crossover rate, using that, we look at the mutation rates and it is clearly seen that 0.01 is the best mutation rate. Thus, in our computations with GA, we use a crossover rate of 0.55 and a mutation rate of 0.01.

5.2.6 The Overall Algorithm

Above, the decisions of the GA model are explained in detail. Basically our genetic algorithm consists of the following four steps:

Step 1. Examine the input data

Step 2. (Randomly) Generate the initial population

Step 3. Interpret each chromosome in the initial population and evaluate the fitness function for each (finding the fittest chromosome in the process)

Step 4. Calculate the population statistics

> **4-a** Find the best fitness
>
> **4-b** Sort the chromosomes according to their fitness
>
> **4-c** Find the elite members of the current generation to make them part of the next generation as well.
>
> **4-d** Sort the chromosomes according to their fitness

Step 5. Until the generation limit is reached or the generation fitness does not increase for the predetermined number of generations:

> **5-a** Select parents from the population using roulette wheel selection,
>
> **5-b** Apply the genetic operators (i.e. crossover the selected parents and mutate the offspring according to the mutation rate),
>
> **5-c** Create the new population,
>
> **5-d** Update the best fitness,
>
> **5-e** Update termination statistics (generation limit and the number of generations the fitness does not increase),

5-e Check for termination, if not return to 4-a

The flowchart of the algorithm is provided in figure 5 below.

The algorithm is coded in C++. The whole code is given in Appendix B in detail.

The pseudo code which is the basis of the code of the algorithm is presented below. The main flow is given as a whole composed of small procedures.

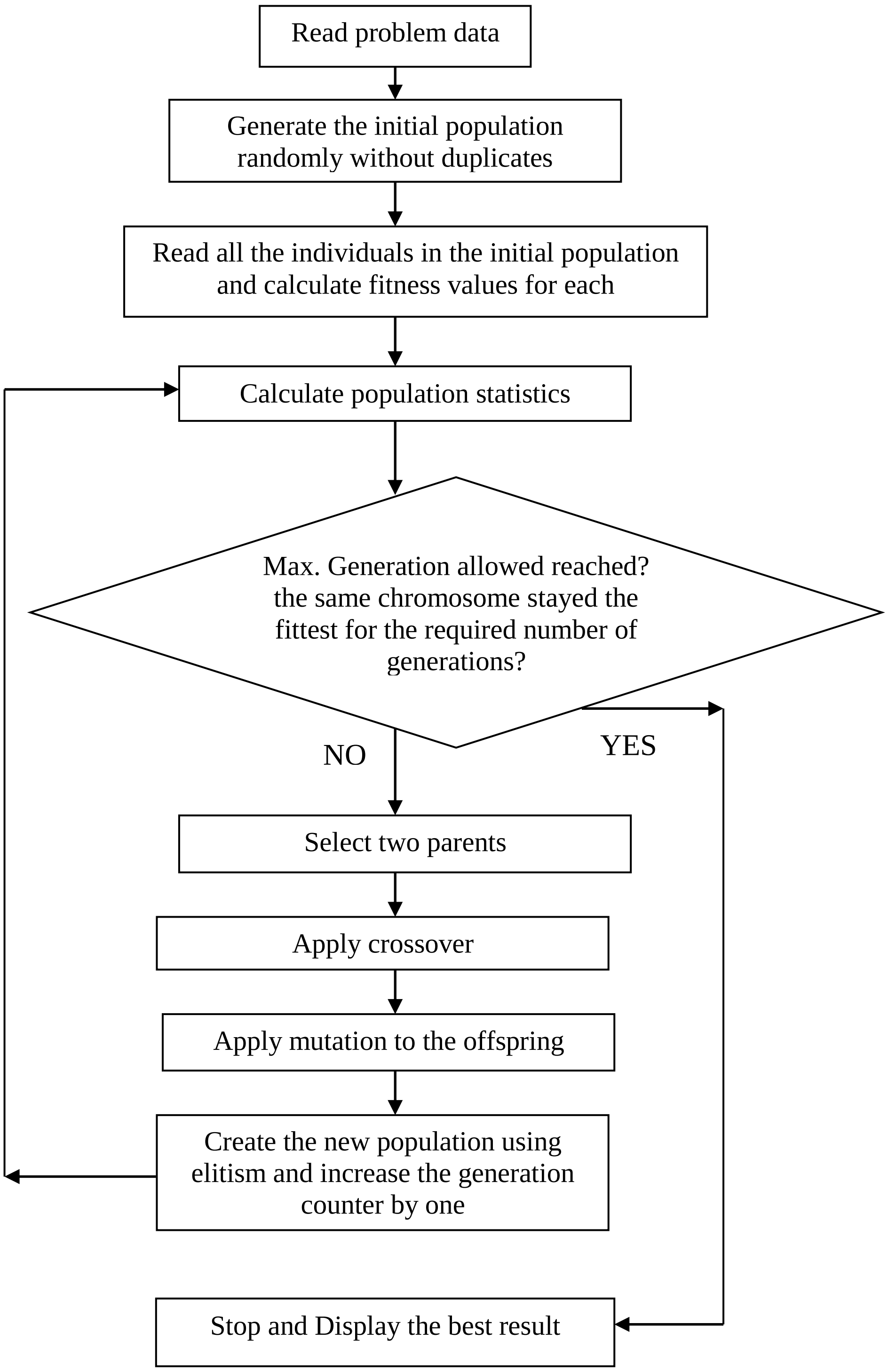

Read problem data
Generate the initial population randomly without duplicates
Read all the individuals in the initial population and calculate fitness values for each
Calculate population statistics
Max. Generation allowed reached? the same chromosome stayed the fittest for the required number of generations?
NO
YES
Select two parents
Apply crossover
Apply mutation to the offspring
Create the new population using elitism and increase the generation counter by one
Stop and Display the best result

Figure 5 Flowchart of the GA algorithm

The Main Body of the Algorithm

```
Procedure SeedRandomNumberGenerator
Procedure ReadData //this is not a procedure, it's directly on the main, but it could be on a procedure,, and it's actually not important where you put it
Procedure CreateRandomPopulation
While ((no chromosome has stayed as the fittest long enough) AND (NumberOfGeneration <= MaximumGeneration)) do
  Procedure AssignFitnessToChromosomesAndFindFittest
  Display the current fittest chromosome
  if (the fittest chromosome prevailed for the required ammount of generations){
    Display "The same chromosome has prevailed as the fittest for GEN_REQUIRED_AS_FITTEST generations!"
    Display Fittest and its fitness
    Set While loop exit condition
  }
  if (NumberOfGeneration <= MaximumGeneration){

    Display "The maximum number of allowed generations has been reached. The current fittest chromosome is the following:"
    Display Fittest and its fitness
    Display "And it has stayed as the fittest chromosome for the last genAsFittest generations"
    Set While loop exit condition
  }
  Procedure Elitism
  While (chromosomes missing to fill population) do
    firstParent = Procedure Roulette
    secondParent = Procedure Roulette
    Procedure Crossover
    Procedure Mutate(firstChild)

    Procedure Mutate(secondChild)
    Add children to population
  endWhile
endWhile
End of Main Body
```

5.3 Discussion of the Results

Here, the results for the proposed genetic algorithm are discussed. The GA is tested against the optimum solution found by total enumeration. We generate the problem instances in order to test our algorithm, since there are no test problems to test it, and researchers generate their own test problems by considering the nature of their problem, to the best of our knowledge. A problem instance is defined by these attributes:

- Number of models to be sequenced
- Sequence length
- Units of each model to be sequenced in the sequence of a certain length (model mix)
- Number of stations making up the assembly line

In our GA approach we assume the cycle time is given and we calculate all statistics according to the given cycle time.

We show how much time it takes to solve these problems by our GA and total enumeration (TE) and if the GA provides the optimum or not. “y” indicates that the solution found is optimal, and leaving the cell empty means we could not test it since the TE could not solve the problem instance in more than the time shown in the “time to solve” column. We present the computational results in Table 17.

Both the Genetic Algorithm and the total enumeration give the same fitness value, however, the sequences obtained may be different. For example, the optimal solutions found for the first problem instance were different even though the fitness value is the same. The sequence of the GA is 3121213121 and the sequence of the total enumeration is 1321211321. These chromosomes are very similar. Only a swap in two places between model 1 and model 3 makes the difference. This is because there are alternative optimal solutions stemming from the nature of the objective function and the chromosomes, but there is no need to try to prevent this.

It is easily seen that the computational time the genetic algorithm takes is much less. Since the time limit by which the genetic algorithm terminates is defined by the user, it could take more time, but in all these problem instances it did not prove necessary to increase the number of generations to run. We see some hard problem instances which the total enumeration could not solve in a long time like 9 days. Hence we could not test the GA for some larger and harder problem instances as seen in the results table, namely problem instances 3, 14, 16, 17, 18, 19.

Table 17 Test of the GA

Problem instance	# of Stations	# of models	Seq length	model mix	time to solve		opt?
					GA	TE	
1	3	3	10	5-3-2	1 sec	1 sec	y
2	5	5	20	8-5-4-2-1	2 sec	6 hours	y
3	3	3	40	20-15-5	2 sec	>2 days	
4	3	3	20	10--6--4	1 sec	25 min	y
5	3	3	15	7-5-3	1 sec	2 sec	y
6	5	3	10	5-3-2	1 sec	1 sec	y
7	5	5	12	4-3-2-2-1	1 sec	5 sec	y
8	4	4	16	6-5-3-2	1 sec	10 min	y
9	4	4	16	4-4-4-4	3 sec	1 day	y
10	3	4	15	4-4-4-3	1 sec	5 sec	y
11	3	4	15	9-3-2-1	1 sec	5 min	y
12	4	4	16	9-4-2-1	1 sec	1 min	y
13	4	4	16	10-3-2-1	1 sec	1min	y
14	3	8	20	4-3-3-2-2-2-2-2	5 sec	>5 days	
15	4	4	20	14-3-2-1	1 sec	7 min	y
16	4	4	20	5-5-5-5	1 sec	>3 days	
17	3	10	20	2-2-2-2-2-2-2-2-2-2	5 sec	> 5 days	
18	5	5	20	4-4-4-4-4	5 sec	> 4 days	
19	3	5	40	12-10-8-6-4	5 sec	> 9 days	
20	30	3	20	10-6-4	1 sec	2 hours	y
21	30	4	20	14-3-2-1	1 sec	7 min	y
22	30	4	20	12-4-2-2	3 sec	8 hours	y
23	30	3	10	5-3-2	1 sec	1 sec	y
24	30	4	16	6-5-3-2	1 sec	10 min	y

Sequence length has a significant effect on the time total enumeration takes to solve. It has an effect on the time the genetic algorithm takes to solve, too, but since it concludes in only a few seconds, it may be called an insignificant effect. Since in the real manufacturing environments, the sequence of models can be simplified to lengths between 10 and 40, the problem instances are conducted using sequence lengths between 10 and 40. We observe that there is a drastic increase in computational times between 10 and 40. Hence, using the genetic algorithm and the like seems mandatory, especially when experimenting with the parameters.

Also the mix of cars in the sequence affects the computation time drastically. When the numbers of models in the mix are very close to each other, like 5-5-5 in a sequence of 15 for 3 models, the computational time increases significantly especially for the total enumeration method. This is simply because the number of combinations the total enumeration method has to check increases with the closer number of cars in the sequence.

Since, in the real world, the number of stations in industrial lines is usually 30 or even more, we test problem instances with long lines, too. The number of stations increases the time the total enumeration takes to solve, too, but not as drastic as the other components of the problem. However, we do not have the same observation with the genetic algorithm.

Overall it is clear that the genetic algorithm is working well and fast. It reaches the optimal sequence no matter how hard the problem instance is and the parameters can easily be adjusted to bigger problem instances.

Finally, we obtain the sequence of the two models in our reference car body shop for the two different assembly line designs: the single line and the mixed-model line.

Table 18 summarizes these results where ‘1’ stands for the cargo model and ‘2’ stands for the combi model. There are 20 cargo models and 13 combi models in a sequence length of 33 cars which is a simpler version of their annual production volumes which are 100,000 and 65,000, respectively for cargo and combi models.

Having obtained the fitness values for the two cases, namely for the lines designed based on single model and mixed model assumptions, respectively, we cannot have a conclusive observation as to which line is better in terms of the sequences’ fitness values. This is somehow an expected result, because we do not consider the objective of leveling the workload at the strategic decision level of mixed-model line deign (robust line design) and the single model line design. but the sequences are different. We provide the sequences obtained for both lines (single and mixed model lines) in Table 18; it can be observed that the sequences obtained for the single and mixed-model lines are all different.

While we are leveling the workload for the stations, one may asks what happens to other possible objective functions. Widely used objective functions like tardiness and earliness doesn’t apply to our situation. Another widely used objective function; makespan can be calculated for our situation however. When we look at the example used in Yu et al. (2005) and sequence it according to our objective criteria and then calculate the makespan, we can see how well it behaves for other objective functions, makespan in this case. Table 19 shows how long it takes for a job on a station (robot).

Table 18 The sequences for the car body shop

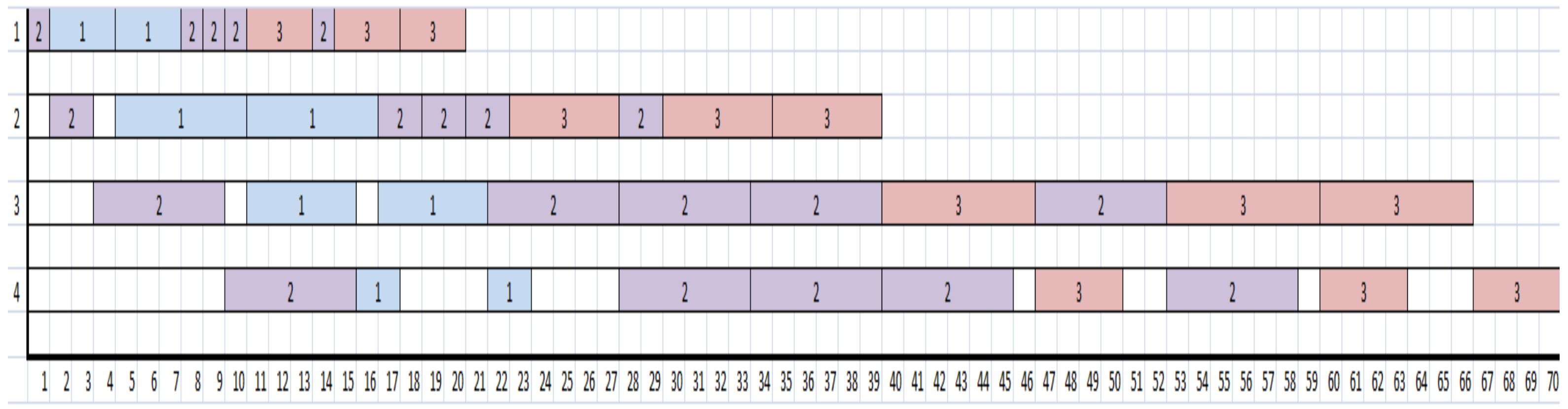
	Fitnesses			
	Mixed-model	Old version	Old version	Mixed-model version
Outside panel sheet	1.20013	1.5771	21212121212112211 2212112121111111	21212112121212121 2121122112111111
Instrument sheet	0.76765	1.48665	21122121212112112 1211212122111111	21212112112121211 2121221122111111
Inside right framework	1.29154	0.47573	21122121212112211 2212112121111111	21212112121121211 2121212112211111
Side panel	1.51638	1.77317	21121121212121221 1221211212111111	21121121121212121 2121212112211111
Dashboard pool	1.54	2.6	21122121212112211 2212112121111111	21212112121212121 2122112211111111
Back stop riser	1.74636	1.45935	21121211212121212 1121221212111111	21212122112121121 1212112211211111
Left shroud	0.76238	0.54761	21121212211211212 1212121212111111	21212121212112122 1122121211111111
Right shroud	0.86	1.39509	21122121212121211 2122121211111111	21212122112121211 2121221121111111
Inside framework back shroud	3.99	3.395	21121212212112121 2121221211111111	21212121212112122 1122121211111111

While we level the workload for the stations in sequencing the several models, we also check to see the performance of our GA in terms of other possible scheduling/sequencing criteria. The commonly used criteria like tardiness and earliness are not relevant in our case. However, the makespan criterion seems more involved in our case. As a result of our search in the literature for GA approaches in mixed model sequencing, we could only have the chance of comparing our GA against the GA study of Yu et al. (2005) that considers the makespan objective in sequencing. The mixed model sequencing problem they address includes 3 product models and 4 stations on the line the assembly times of which are given in Table 19. We use this test problem and obtain a sequence with our GA method, that is with the workload leveling criterion. The sequence we thus obtain (Figure 6) is found to have a makespan of 70 time units, while the GA approach developed by Yu et al. (2005) with the makespan criterion ends up with a makespan of 67 time units.

Table 19 The assembly time for three products on four robots

Product	**Assembly time (unit time)**			
	Robot 1	**Robot 2**	**Robot 3**	**Robot 4**
1	3	6	5	2
2	1	2	6	6
3	3	5	7	4

Figure 6 Gantt Chart for the Makespan test problem with the best sequence 2-1-1-2-2-2-3-2-3-3



CHAPTER 6

CONCLUSION AND FURTHER RESEARCH ISSUES

In this study, we address two problems: assembly line design which is a strategical decision and mixed-model sequencing which is an operational decision. For the assembly line design problem we propose a robust optimization model based on integer programming which is an extension of a former approach. We extend the former approach that is proposed for the single-model line environment to a mixed-model environment. In our robust optimization model we try to minimize the maximum regret over all models of the car, considering the total of investment and operating costs. We then design all the lines for the car body shop with reference to a local car manufacturer.

After designing the mixed model lines in the car body shop, we address the operational problem of sequencing on the lines which have a mixed-model nature. Since the sequencing problem is an NP-hard problem and hence hard to solve by mathematical modeling, we intend to solve it by a genetic algorithm which is fast and eligible for sequencing problems. We develop a genetic algorithm and then search for its performance through several test problems. We compare the solutions of the GA against the optimum solutions obtained by total enumeration.

From the computational work, we also see how the problem size affects the computational time of a run. The problem size is determined by the sequence length, combination of models of cars and number of stations. We conclude that our GA works well and fast and can be used to find the optimal sequence for any mixed-model sequencing problem with the objective of workload smoothing over the stations over time.

Further analyses can be made on the robust optimization model to see how well it reacts to changes in the annual demand and the changes in total cost can be observed in the former single line approach and the new mixed-model line approach we propose. Similarly, the robust optimization model can be extended so as to take into account the dynamic demand over the lifecycle of a car, that is increasing annual demand during the first years, and then stable demand for some time, and decaying demand during the last years of the life cycle.

Our GA can be tested with other fitness functions. Also, multi objective fitness functions can be tried since they are getting increasingly more popular day by day.

Paint shop constraints can be taken into account in the mixed-model sequencing algorithm in the car body shop which results in the so-called ‘car sequencing problem’ that has got a systemic approach for the model sequencing in the body shop.

REFERENCES

Akgündüz, O., Tunalı, S., 2011: A review of the current applications of genetic algorithms in mixed-model assembly line sequencing, *International Journal of Production Research* **49:15**, 4483-4503

Amen, M., 1998. Heuristic methods for cost-oriented assembly line balancing: A survey, *Int. J. Production Economics* **68** 1-14

Amen, M., 2000. An exact method for cost-oriented assembly line balancing, *International Journal of Production Economics* **64** 187-195

Barutçuoğlu, A., Kırca, Ö., Meral, S., 2011. Gövde Atölyesi Otomasyon Seviyesi Optimizasyonu, ODTÜ Endüstri Mühendisliği Bölümü

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

Bradley, J., 2002. Using Product-Mix Flexibility to Implement a Made-To-Order Assembly Line

Bukchin, J. 1998. A comparative study of performance measures for throughput of a mixed model assembly line in a JIT environment, *International Journal of Production Research* **36:10**, 2669-2685

Deep K. and Mebrahtu H., 2011. New Variations of Order Crossover for Travelling Salesman Problem, *International Journal of Combinatorial Optimization Problems and Informatics* **2:1** 2-13

Emde, S., Boysen, N., Scholl, A., 2008. Balancing mixed-model assembly lines: A computational evaluation of objectives to smoothen workload, *Working and Discussion Paper Series School of Economics and Business Administration Friedrich-Schiller-University* Jena ISSN 1864-3108

Gao, J., Sun, L., Wang, L., Gen, M., 2009. An efficient approach for type II robotic assembly line balancing problems, *Computers & Industrial Engineering* **56** 1065–1080

Ghosh, S., Gagnon, R., 1989. A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems, *International Journal of Production Research* **27:4**, 637-670

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

Goldberg, D.E., 1989. Sizing Populations for Serial and Paralel Genetic Algorithms in *Proceedings of the Third International Conferene on Genetic Algorithms by Schaffer, J. D.*, Morgan Kaufmann Publishers, San Mateo, 70-79.

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

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

Guo Z., Wong W., Leung S., Fan J. and Chan S., 2006. A genetic-algorithm-based optimization model for scheduling flexible assembly lines, *International Journal of Advanced Manufacturing Technologies* **36**, 156–168

Haq A., Jayaprakash J., Rengarajan K., 2004. A hybrid genetic algorithm approach to mixed-model assembly line balancing, *International Journal of Advanced Manufactoring Technologies* **28** 337–341

Holland, J.H., 1975. Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor, MI.

Hwang, R., Katayama, H., 2010. Integrated procedure of balancing and sequencing for mixed-model assembly lines: a multi-objective evolutionary approach, I*nternational Journal of Production Research* **48:21**, 6417–6441

Kim, H., Park, S., 2007. A strong cutting plane algorithm for the robotic assembly line balancing problem, *International Journal of Production Research* **33:8**, 2311-2323

Kis, T. 2004. A On the complexity of the car sequencing problem, *Operation Reserach Letters* **32:4**, 331-335

Kleeman M.P. and Lamont G.B., 2007. Scheduling of Flow-Shop, Job-Shop, and Combined Scheduling Problems using MOEAs with Fixed and Variable Length Chromosomes, *Studies in Computational Intelligence (SCI)* **49**, 49–99

Levitin G., Rubinovitz J., Shnits B., 2004. A genetic algorithm for robotic assembly line balancing, *European Journal of Operational Research*

Nicosia, G., Pacciarelli, D., Pacifici, A., 2002. Optimally balancing assembly lines with different workstations, *Discrete Applied Mathematics* **118** 99–113

Norman, B., Bean, J., 1994. A genetic algorithm code for scheduling problems: serial computing version

Rajput, F.A., Othman, Z., Suhail, A., 2010. Comparative Study between Mixed Model Assembly Line and Flexible Assembly Line Based on Cost Minimization Approach, *Proceedings of the 2010 International Conference on Industrial Engineering and Operations Management Dhaka, Bangladesh*

Rekiek, B., Doigui, A., Delchambre, A., Bratcu, A., 2002. State of Art of Optimization Methods for Assembly Line Design, *Annual Reviews in Control* **26** 163-174

Smith, J. and Vavak, F., 1999. Replacement Strategies in Steady State Genetic Algorithms : Static Environments in *Foundations of Genetic Algorithms-5 by Banzhaf W.*, Morgan Kaufmann Publishers, San Francisco,219-233.

Spieckermann S., Gutenschwager K. and Vob S., 2004. A sequential ordering problem in automotive paint shops, Int. J. Prod. Res., **42:9**, 1865–1878

Thomopoulos, N., 1967. Line Balancing-Sequencing for Mixed-Model Assembly, *Management Science* **14:2**, 59-75

Wang B., Rao Y., Shao X., and Wang M., 2008. Scheduling Mixed-Model Assembly Lines with Cost Objectives by a Hybrid Algorithm, ICIRA 2008, Part II, LNAI 5315, 378–387

Yu, J., Yin, Y. and Chen, Z., 2005. Scheduling of an assembly line with a multi-objective genetic algorithm, *International Journal of Advanced Manufacturing Technologies* **28**, 551–555

APPENDIX A

THE SOLUTION SHEET OF THE DASHBOARD POOL

Table A.1 The Solution Excel Page Costs and Important Information

max regret	2363326,281
Total investment and transformation cost(€)	5059433,47
Total investment and transformation cost(€)combi	3940308,281
Total investment and transformation cost(€)cargo	3678134,639
Total investment cost(€)	1066000
Transformation cost combi(€)	2744333,281
Transformation cost cargo(€)	2482159,639
Total transformation cost(€)	3993433,47
Total Cost of Space(€)	129975
Total Used Space(m2)	173,3
Automation %	74,83221577
Number of Robotic stations	4
Number of Manual stations	1
Number of stations	5
Number of Robots	4
Number of workers	2
Number of workers + robots	6

Table A.2 The Solution Excel Page Inputs Reminder

Daily working hours	21
Years of production(years)	8
Total yearly production	165000
Total yearly production-combi	100000
Total yearly production-cargo	65000
Cost of labor (€/h)	7,5

Table A.3 The Solution Excel Page Important Information of Stations

	1	2	3	
	M2	R15	R2	
Number of workers	2	1	1	
Time used (sec)	159,56672	18	69,4	
Time without added value (sec)	104,36672	18	46	
Time of tasks(sec)	55,2		23,4	
Usable time (sec)	159,6672	73,4832	73,4832	
Station utility (%)	99,9370691	24,49539487	94,44335576	
Conveyor worker			0,132275132	
Work done in terms of bolts	13,5		9	
stations	1	3	4	5
tasks	M2	R2	R35	R17
8A01	1			
8V01	1			
8T01	1			
8T02	1			
8T03		1		
8T04		1		
8T05		1		
8D01		1		
8K01			1	
8K02			1	
8M01				1

Table A.4 The Solution Excel Page Task Assignments

stations	1	2	3	4	5
tasks	M2	R15	R2	R35	R17
8A02		1			
8V02	1				
8T06	1				
8T07			1		
8T08			1		
8T09			1		
8D02		1			
8K03				1	
8K04				1	
8M02					1
8B02		1			
stations	4				
tasks	R35				
8S01	10				
stations	3	4			
tasks	R2	R35			
8S02	1,775304348	8,224695652			

APPENDIX B

THE CODE OF THE GA

```
#include <string>
#include <iostream>
#include <math.h>

using std::string;
using std::cout;
using std::cin;
using std::endl;

#define CROSSOVER_RATE            0.7
#define MUTATION_RATE             0.01
#define POP_SIZE                  200        //must be an even number
#define ELITE_SIZE                      20       //must be smaller than POP_SIZE, and an
EVEN number
#define MAX_ALLOWABLE_GENERATIONS   400
#define GEN_REQUIRED_AS_FITTEST     50

//returns a float between 0 & 1
#define RANDOM_NUM      ((float)rand()/(RAND_MAX+1))

//-------------------------------------------------------------------------------------
```

```
//
//   define a data structure which will define a chromosome
//
//-------------------------------------------------------------------------------------
struct chromo_typ
{
  //the binary bit string is held in a std::string
  string   bits;
  float    fitness;

  // if I don't give values, it creates a default chromosome with empty bits and zero
fitness
  chromo_typ(): bits(""), fitness(0.0f){};
  // otherwise, I assing the given values to it
  chromo_typ(string bts, float ftns): bits(bts), fitness(ftns){}
};


////////////////////////////////prototypes////////////////////////////////////////////
string  GetRandomBits(int length, int carsAmmount, int* carProportions);
float    AssignFitness(string bits, int chromoLength, int stationsAmmount, float*
AverageTimes, float **CarsTimeByStation);
string  Roulette(float total_fitness, chromo_typ* Population);
void    Mutate(string &bits);
void    Crossover(string &offspring1, string &offspring2, int chromoLength);
void    Elitism(chromo_typ* Elite, chromo_typ* Population, int eliteSize);
int        compareChromo(const void* elem1, const void* elem2); //chromosome
comparison function for qsort, based on fitness
```

```
float **newMatrix(int rows, int cols);
void deleteMatrix(float **matrix, int rows);
void printBits(string &bits, int chromoLenght);

//------------------------------main------------------------------------------------
//
//---------------------------------------------------------------------------------
int main()
{
   //seed the random number generator
   srand((int)time(NULL));

   //chromosome lengh
   int chromoLength;
   //storage for our population of chromosomes.
   chromo_typ Population[POP_SIZE];
   // ammount of stations in the factory
   int stationsAmmount=0;
   //ammount of car 'models'
   int carsAmmount=0;
   //storage for the car proportions in the chromosome
   int *carProportions;
   //storage for time taken by each station to work on each type of car
   float **CarsTimePerStation;
   //storage for the average desired time per station
   float *AverageTimeByStation;
   // best solution of the last generation
   chromo_typ Fittest;
```

```
// number of generations with the same fittest chromosome
int genAsFittest = 0;
// temporary Integer... sorry, but I need it
int tempInt;

cout << "\nInput the desired sequence lenght: ";
cin >> chromoLength;

while (!(carsAmmount*stationsAmmount)){
   //get ammount of stations from user
   cout << "\nInput the desired stations ammount (greater than 0): ";
   cin >> stationsAmmount;
   cout << endl;

   //get ammount of cars
   cout << "\nInput the desired ammount of car models (greater than 0): ";
   cin >> carsAmmount;
   cout << endl;
};

//allocate memory for the car proportions
carProportions = new int[carsAmmount];

for (int i=0; i < carsAmmount; i++)
   carProportions[i] = 0;

//get car proportions
tempInt = 0;
```

```
    for (int i=0; i < carsAmmount && tempInt < chromoLength; i++)
    {
        cout << "\nInput the ammount of cars from model " << (i + 1) << " in the
sequence: ";
        cin >> carProportions[i];
        if ((tempInt + carProportions[i]) > chromoLength)
          {
              cout << "\nGood work! You managed to go beyond the sequence lenght. ";
              carProportions[i] = chromoLength - tempInt;
              cout << "The ammount of cars for model " << (i + 1) << " has been set to
";
              cout << carProportions[i] << ". The rest of the models' ammount will be set
to 0.";
              cout << endl;
          }
        tempInt += carProportions[i];
    }

    if (tempInt < chromoLength){
       cout << "\nThe car proportions that you entered are not enough to cover the
sequence lenght.";
       cout << "\nCar proportion for model " << carsAmmount << " will be set to " <<
(chromoLength - tempInt + carProportions[carsAmmount-1]);
       carProportions[carsAmmount-1] = chromoLength - tempInt +
       carProportions[carsAmmount-1];
    }

    //allocate memory for the time taken by each station to
```

```
   //work on each type of car
   CarsTimePerStation = newMatrix(carsAmmount, stationsAmmount);

   //allocate memory for the average times per station
   AverageTimeByStation = new float[stationsAmmount];

   //get times from the user
   for (int i=0; i < stationsAmmount; i++)
   {
      cout << "\nInput the average desired time for station " << (i + 1) << ": ";
      cin >> AverageTimeByStation[i];
   }

   for (int i=0; i < carsAmmount; i++)
   {
      for (int j=0; j < stationsAmmount; j++)
      {
         cout << "\nInput time taken by station " << (j + 1) << " to work on car type "
<< (i + 1) << ": ";
         cin >> CarsTimePerStation[i][j];
         cout << endl;
      }
   }

    //first create a random population, all with zero fitness.
    for (int i=0; i<POP_SIZE; i++)
    {
```

```
    Population[i].bits = GetRandomBits(chromoLength, carsAmmount,
carProportions);
    Population[i].fitness = 0.0f;
  }

  int GenerationsRequiredToFindASolution = 0;

  //we will set this flag if the best solution is the same for
GEN_REQUIRED_AS_FITTEST generations
  bool bFound = false;

  //enter the main GA loop
  while(!bFound)
  {
    //this is used during roulette wheel sampling
    float TotalFitness = 0.0f;

    // test and update the fitness of every chromosome in the population
    // AND find the fittest, in the meantime
    for (int i=0; i<POP_SIZE; i++)
    {
      // We assign the fitness to each chromosome
      Population[i].fitness = AssignFitness(Population[i].bits, chromoLength,
stationsAmmount, AverageTimeByStation, CarsTimePerStation);

      if (Population[i].fitness > Fittest.fitness)
      {
        Fittest.bits = Population[i].bits;
```

```
          Fittest.fitness = Population[i].fitness;
          genAsFittest = 0;
        }

        TotalFitness += Population[i].fitness;
      }

      // a new generation has just passed, so we increase genAsFittest and
      // GenerationsRequiredToFindASolution, and check to see if the Fittest
      // is still the same for the last GEN_REQUIRED_AS_FITTEST generations
      // or the MAX_ALLOWABLE_GENERATIONS threshold has been reached
      ++GenerationsRequiredToFindASolution;

      //Let's show the fittest in each round
      cout << "Best solution in round " << GenerationsRequiredToFindASolution <<
": ";
      for (int i=0; i < chromoLength; i++)
         cout << (int)Fittest.bits.at(i) + 1;
      cout << " | Fitness: " << Fittest.fitness << endl;

      if ((++genAsFittest) == GEN_REQUIRED_AS_FITTEST)
      {
         cout << endl << "The same chromosome has prevailed as the fittest for " <<
GEN_REQUIRED_AS_FITTEST << " generations!" << endl;

         cout << "It took " << GenerationsRequiredToFindASolution << " generations
to get to the following solution:" << endl << endl;
```

```
        for (int i=0; i < chromoLength; i++)
            cout << (int)Fittest.bits.at(i) + 1;

        cout << endl << endl << "And its fitness is: ";
        cout << Fittest.fitness <<endl;

        bFound = true;
    }

    if                    (GenerationsRequiredToFindASolution                    >
MAX_ALLOWABLE_GENERATIONS)
    {
        cout << "The maximum number of allowed generations has been reached. The
current fittest chromosome is the following:" << endl;

        for (int i=0; i < chromoLength; i++)
            cout << (int)Fittest.bits.at(i) + 1;

        cout << endl << endl << "Its fitness is: ";
        cout << Fittest.fitness << endl;

        cout << "And it has stayed as the fittest chromosome for the last ";
        cout << genAsFittest << " generation";
        cout << (genAsFittest > 1 ? "s." : ".") << endl;

        bFound = true;
    }
    // create a new population by first implementing elitism, and then
    // selecting two parents at a time and creating offspring by applying
    // crossover and mutation. Do this until the desired number of offspring
    // have been created.

    //define some temporary storage for the new population we are about to create
    chromo_typ temp[POP_SIZE];

    // IMPLEMENT ELITISM HERE TO GET THE BEST SOLUTIONS TO THE
NEXT GENERATION
    // ELITE_SIZE is the ammount of chromosomes on the elite.
    Elitism(temp, Population, ELITE_SIZE);

    //loop until we have created POP_SIZE - ELITE_SIZE new chromosomes
    int cPop = ELITE_SIZE;

    while (cPop < POP_SIZE)
    {
      // we are going to create the rest of the new population by grabbing
      // members of the old population two at a time via roulette wheel selection.
      string offspring1 = Roulette(TotalFitness, Population);
      string offspring2 = Roulette(TotalFitness, Population);

      //add crossover dependent on the crossover rate
      Crossover(offspring1, offspring2, chromoLength);

      //now mutate dependent on the mutation rate
      Mutate(offspring1);
      Mutate(offspring2);

      //add these offspring to the new population. (assigning zero as their
      //fitness scores)
      temp[cPop++] = chromo_typ(offspring1, 0.0f);
      temp[cPop++] = chromo_typ(offspring2, 0.0f);

    }

    //copy temp population into main population array
    for (int i=0; i<POP_SIZE; i++)
    {
      Population[i] = temp[i];
    }
  }

cout << "\n\n\n";

/* DEBUG: show all list
for (int i=0; i<POP_SIZE; i++)
{
   for (int j=0; j < chromoLength; j++)
        cout << (int)Population[i].bits.at(j);
   cout << " " << Population[i].fitness << " | ";
}
cout << "\n\n\n";
*/
```

```
    // just to see the results
    string pepe;
    cout << "Type a letter and press <ENTER>" << endl;
    cin >> pepe;

    //Memory cleaning
    delete [] AverageTimeByStation;
    delete [] carProportions;
    deleteMatrix(CarsTimePerStation, carsAmmount);
    return 0;
}



//--------------------------------newMatrix--------------------------
//
// This function dynamically allocates a matrix
//
//------------------------------------------------------------------
float **newMatrix(int rows, int cols){
    float** storage = new float*[rows];
    if (rows)
    {
        storage[0] = new float[rows * cols];
        for (int i = 1; i < rows; ++i)
            storage[i] = storage[0] + i * cols;
    }
    return storage;
};
```

```
//-----------------------------deleteMatrix--------------------------
//
// Take a guess...
//
//-------------------------------------------------------------------
void deleteMatrix(float **matrix, int rows){
   if (rows) delete [] matrix[0];
   delete [] matrix;
};

//--------------------------------printBits--------------------------
//
// Take another guess...
//
//-------------------------------------------------------------------
void printBits(string &bits, int chromoLength){
  for (int i=0; i < chromoLength; i++)
      cout << (int)bits.at(i);
};



//--------------------------------GetRandomBits----------------------------------------
//
//   This function returns a string of random As and Bs of the desired length.
//   We want the proportion A:B to be MAX_A:MAX_B
//-------------------------------------------------------------------------------------
string GetRandomBits(int length, int carsAmmount, int* carProportions)
```

```
{
  //we're gonna define 'model segment' as the subinterval of [0..1]
  //which will correspond to a given i model, in the following way:
  //the model segment for model i, would be
[(i/carsAmmount)..((i+1)/carsAmmount)]
  string bits;
  int *counters;
  float randomNum;
  //temporary Float to find the model segment. It's a float to avoid the integer
  //division in the while (below), since that would always result in 0 otherwise
  float tempFloat;
  // temporary Integer for the indexes, since C doesn't like float indexes, and
  // I don't wanna be casting it all the time
  int tempInt;

  counters =  new int[carsAmmount];

  for (int i=0; i<carsAmmount; i++)
     counters[i] = 0;

  for (int i=0; i<length; i++)
  {
     //sorry, but I need to use the random a couple of times, and if I call
     //RANDOM_NUM every time, it will be a different number, so I have to store it.
     randomNum = RANDOM_NUM;
     //no sense to start from 0, since it will be always smaller
     tempFloat = 1.0f;
     //search the model segment
```

```
        while ((tempFloat++/carsAmmount) < randomNum);
        //the index we look for is the low end of the segment
        //so we need to decrease the index by one. Also, we need
        //to decrease it by one more, since the indexes of an array
        //start with 0, and not with 1
        tempInt = (int) tempFloat - 2;

        //if I hasn't exceded the proportion, I still can add
        //that car model, if not, I need to randomly pick the
        //car model for this chromosome again
        if (counters[tempInt] < carProportions[tempInt])
        {
            bits += tempInt;
            counters[tempInt]++;
        }
        else
        {
            i--;
        }
    };

    //some memory cleaning
    delete [] counters;

    return bits;
}

//---------------------------------AssignFitness-------------------------------------
```

```
//
//   given a string of bits, average expected times by station, and times needed
//   to work on each car by station, this function will calculate the fitness
//   score for a the given chromosome
//--------------------------------------------------------------------------------
float AssignFitness(string bits, int chromoLength, int stationsAmmount, float*
AverageTimes, float** CarsTimePerStation)
{
   float fitScore = 0;
   float **Utility;

   Utility = newMatrix(stationsAmmount,chromoLength);

   /////////////////////////////
   // This should go into a function (maybe called calculateUtility) but...
   // what can we do? There's no time for tidiness this round :)

   // I calculate the first column separatedly to use that after to calculate
   // the utilities recursivelly
   // This is done just to avoid intermediate calculations in the next loop
   for (int i = 0; i < stationsAmmount; i++){
      // I'm multiplying by 1.0 just to make sure Utility get's a float
      Utility[i][0]     =     (CarsTimePerStation[((int)bits.at(0))][i]     *     1.0f)     /
AverageTimes[i];
   };

   for (int i = 0; i < stationsAmmount; i++){
      for (int j = 1; j < chromoLength; j++){
            // As C++ indexes start with 0, I will use j and j+1 instead of j-1 and j in the
formula
            Utility[i][j]  =  (Utility[i][j-1]  *  j  /  (j+1))  +
((CarsTimePerStation[(int)bits.at(j)][i] * 1.0f) / (AverageTimes[i] * (j+1)));
        }
    }
    //
    ////////////////////

    // Now, the fitness would be the summatory of all the utilities differences
    for (int i = 0; i < stationsAmmount; i++){
        for (int j = 1; j < chromoLength; j++){
            fitScore += fabs(Utility[i][j] - Utility[i][j-1]);

        }
    }

    // some cleaning
    deleteMatrix(Utility, stationsAmmount);

    return fitScore;
}



//-----------------------------------Mutate-------------------------------------
//
//   Mutates a chromosome's bits dependent on the MUTATION_RATE
//-----------------------------------------------------------------------------------
```

```
void Mutate(string &bits)
{
  size_t found;
  //I don't want to calculate this each time I enter the for loop below, and
  //memory is an elephant :)
  int bitsLength = bits.length();

  for (int i=0; i<bitsLength; i++)
  {
    if (RANDOM_NUM < MUTATION_RATE)
    {
      int j = 0;
      int swapBit;
      while (bits.at(j++) == bits.at(i));
      swapBit = bits.at(--j);
      bits.at(j) = bits.at(i);
      bits.at(i) = swapBit;
    }
  }
};

//--------------------------------- Crossover --------------------------------------
//
// Dependent on the CROSSOVER_RATE this function selects random points along
the
// lenghth of the chromosomes and applies an ADAPTED ordered crossover (OX)
on the parents.
//---------------------------------------------------------------------------------
```

```
void Crossover(string &offspring1, string &offspring2, int chromoLength)
{
  //dependent on the crossover rate
  if (RANDOM_NUM < CROSSOVER_RATE)
  {
    //create random crossover points
    int crossover1 = (int) (RANDOM_NUM * chromoLength);
    int crossover2 = (int) (RANDOM_NUM * chromoLength);

    if (crossover2 < crossover1)
    //swap crossover points with XOR
    {
           crossover1 = crossover1 ^ crossover2;
           crossover2 = crossover1 ^ crossover2;
           crossover1 = crossover1 ^ crossover2;
    }

    //take slices
    string middle1 = offspring1.substr(crossover1, crossover2 - crossover1);
    string middle2 = offspring2.substr(crossover1, crossover2 - crossover1);

    string tempoff1, tempoff2;
    string midtemp1 = middle1, midtemp2 = middle2;

    // I'll go through the parent2, checking in order if the gene is in the slice
    // picked from parent1. If so, I will avoid storing that gene, otherwise it
    // will be added to the genes that will form part of offspring1
    tempoff1.clear();
```

```
size_t found;
for (int i = 0; i < chromoLength; i++)
{
   found = midtemp1.find(offspring2[i]);
     if (found!=string::npos)
     {
       midtemp1.erase(found,1);
     }
     else
     {
       tempoff1 += offspring2[i];
     }
}

// idem
tempoff2.clear();
for (int i = 0; i < chromoLength; i++)
{
   found = midtemp2.find(offspring1[i]);
     if (found!=string::npos)
     {
       midtemp2.erase(found,1);
     }
     else
     {
       tempoff2 += offspring1[i];
     }
}
```

```
  offspring1 = tempoff2.substr(0, crossover1) + middle2 + tempoff2.substr(crossover1);
  offspring2 = tempoff1.substr(0, crossover1) + middle1 + tempoff1.substr(crossover1);
 }
}


//--------------------------------Roulette-----------------------------------------
//
//   selects a chromosome from the population via roulette wheel selection
//---------------------------------------------------------------------------------
string Roulette(float total_fitness, chromo_typ* Population)
{
  //generate a random number between 0 & total fitness count
  float Slice = (float)(RANDOM_NUM * total_fitness);

  //go through the chromosones adding up the fitness so far
  float FitnessSoFar = 0.0f;

  for (int i=0; i<POP_SIZE; i++)
  {
    FitnessSoFar += Population[i].fitness;

    //if the fitness so far > random number return the chromo at this point
    if (FitnessSoFar >= Slice)
```

```
        return Population[i].bits;
   }

   return "";
}

//-------------------------------Elitism-----------------------------------------
//
//   selects the fitest chromosomes from the Population
//--------------------------------------------------------------------------------
void Elitism(chromo_typ* Elite, chromo_typ* Population, int eliteSize){
     qsort(Population, POP_SIZE, sizeof(chromo_typ), compareChromo);
     for (int i = 0; i < eliteSize; i++)
         Elite[i] = chromo_typ(Population[i].bits, Population[i].fitness);
}

//-------------------------------compareChromo-----------------------------------------
//
//   chromosome comparison function for qsort, based on fitness
//--------------------------------------------------------------------------------
int compareChromo(const void* elem1, const void* elem2){
    if ( ((chromo_typ*)elem1)->fitness > ((chromo_typ*)elem2)->fitness )
       return -1;
    else
       return 1;
    }
```