ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΒΙΟΜΗΧΑΝΙΑΣ

Μεταπτυχιακή Εργασία

ΧΡΟΝΟΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΒΑΡΔΙΩΝ ΔΥΝΑΜΟΛΟΓΙΟΥ ΟΔΗΓΩΝ ΑΜΑΞΟΣΤΟΙΧΕΙΩΝ ΜΕ ΧΡΗΣΗ ΓΡΑΜΜΙΚΟΥ ΑΚΕΡΑΙΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

υπό

ΣΚΙΑΔΟΠΟΥΛΟΣ ΕΥΑΓΓΕΛΟΣ

Διπλωματούχου Μηχανολόγου Μηχανικού Πολυτεχνικής Σχολής ΑΠΘ, 2009

Υπεβλήθη για την εκπλήρωση μέρους των

απαιτήσεων για την απόκτηση του

Μεταπτυχιακού Διπλώματος Ειδίκευσης

2013

Η έγκριση της μεταπτυχιακής εργασίας από το Τμήμα Μηχανολόγων Μηχανικών Βιομηχανίας της Πολυτεχνικής Σχολής του Πανεπιστημίου Θεσσαλίας δεν υποδηλώνει αποδοχή των απόψεων του συγγραφέα (Ν. 5343/32 αρ. 202 παρ. 2).

Εγκρίθηκε από τα Μέλη της Τριμελούς Εξεταστικής Επιτροπής:

Πρώτος Εξεταστής (Επιβλέπων) — Δρ. Αθανάσιος Ζηλιασκόπουλος
Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών,
Πανεπιστήμιο Θεσσαλίας

Δεύτερος Εξεταστής — Δρ. Δημήτρης Παντελής
Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών,
Πανεπιστήμιο Θεσσαλίας

Τρίτος Εξεταστής — Δρ. Γεώργιος Σαχαρίδης
Λέκτορας, Τμήμα Μηχανολόγων Μηχανικών,
Πανεπιστήμιο Θεσσαλίας

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στο τομέα Βελτιστοποίησης Συστημάτων Παραγωγής / Μεταφορών της σχολής Μηχανολόγων Μηχανικών του Πανεπιστημίου Θεσσαλίας στο πλαίσιο εμβάθυνσης του μεταπτυχιακού μαθήματος «Βελτιστοποίηση και Ροές σε δίκτυα με εφαρμογές σε συστήματα Logistics» υπό την επίβλεψη του Καθηγητή Δρ. Ζηλιασκόπουλου Αθανάσιου.

Με την ολοκλήρωση της διπλωματικής μου εργασίας, απέκτησα Μεταπτυχιακό Δίπλωμα Ειδίκευσης στις «Σύγχρονες Μεθόδους Σχεδιασμού και Ανάλυσης στη Βιομηχανία». Κρίνω λοιπόν απαραίτητο να ευχαριστήσω θερμά τους ανθρώπους που με στήριξαν και με βοήθησαν για την ολοκλήρωση των σπουδών μου.

Αρχικά θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου Δρ. Ζηλιασκόπουλο Αθανάσιο για την εμπιστοσύνη που μου έδειξε, όσον αφορά την ανάθεση του συγκεκριμένου θέματος. Θα ήθελα επίσης να ευχαριστήσω θερμά το κ. Λόη Αθάνασιο - επιστημονικό συνεργάτη το εργαστηρίου βελτιστοποίησης συστημάτων του ΤΜΜ για το ενδιαφέρον και τη συνεχή στήριξη και βοήθεια σε κάθε βήμα αυτής μου της προσπάθειας. Επίσης, είμαι ευγνώμων στα υπόλοιπα μέλη της εξεταστικής επιτροπής της μεταπτυχιακής εργασίας μου, Δρ. Σαχαρίδης Γεώργιος, Δρ. Παντελής Δημήτριος, για την προσεκτική ανάγνωση της εργασίας μου και για τις πολύτιμες υποδείξεις τους.

Τέλος θα ήθελα να ευχαριστήσω τους συμφοιτητές και φίλους μου Μπεσλεμέ Αντώνη, Παπαευσταθίου Χαράλαμπο και Δημήτρη Διαμαντή για τη πολύτιμη βοήθεια τους κατά τη διάρκεια των σπουδών μου, μα πάνω από όλα ευχαριστώ τους γονείς μου Κωνσταντίνο και Παρασκευή για όλα τα εφόδια που μου προσέφεραν και τους αφιερώνω τη διπλωματική μου εργασία.

Σκιαδόπουλος Ευάγγελος

ΧΡΟΝΟΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΒΑΡΔΙΩΝ ΔΥΝΑΜΟΛΟΓΙΟΥ ΟΔΗΓΩΝ ΑΜΑΞΟΣΤΟΙΧΕΙΩΝ ΜΕ ΧΡΗΣΗ ΓΡΑΜΜΙΚΟΥ ΑΚΕΡΑΙΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

ΣΚΙΑΔΟΠΟΥΛΟΣ ΕΥΑΓΓΕΛΟΣ

Πανεπιστήμιο Θεσσαλίας, Τμήμα Μηχανολόγων Μηχανικών Βιομηχανίας, 2013

Επιβλέπων Καθηγητής: Δρ. Ζηλιασκόπουλος Αθανάσιος, Καθηγητής Βελτιστοποίησης Συστημάτων Παραγωγής / Μεταφορών

Περίληψη

Η παρούσα μελέτη αφορά το χρονοπρογραμματισμό (rostering) μηχανοδηγών από ένα συγκεκριμένο μηχανοστάσιο σε βάρδιες. Η επίλυση του προβλήματος έγινε με τη χρήση ενός καινούργιου αλγόριθμου βελτιστοποίησης γραμμικού ακέραιου προγραμματισμού. Σκοπός μας ήταν η δημιουργία ενός ισορροπημένου, επταήμερου κυκλικού προγράμματος ικανοποιώντας τους εργασιακούς κανονισμούς και τις πολιτικές της Τραινοσε Α.Ε.

Αρχικά περιγράφουμε το πρόβλημα και όλες της παραμέτρους που το απαρτίζουν. Στη συνέχεια γίνεται εκτενής αναφορά στη βιβλιογραφία και πως αντιμετώπισαν, σε διάφορες χώρες, το συγκεκριμένο πρόβλημα στη πράξη.

Αναλύουμε κάποια σημαντικά μοντέλα βελτιστοποίησης και τις αδυναμίες τους και παρουσιάζουμε μία μαθηματική προσέγγιση του προβλήματος. Παράλληλα παρουσιάζονται και εξηγούνται, η αντικειμενική συνάρτηση και οι περιορισμοί του προβλήματος και επιλύονται αριθμητικά παραδείγματα.

Ο κώδικας που χρησιμοποιήθηκε για την υλοποίηση του αλγόριθμου, γράφτηκε σε γλώσσα προγραμματισμού C++ με τη βοήθεια του λογισμικού IBM ILOG, και τα απαραίτητα αριθμητικά πειράματα εκτελέστηκαν σε έναν από τους server του πανεπιστημίου.

Η μελέτη ολοκληρώνεται με τη συγγραφή των συμπερασμάτων και αξιολογήσεων και γίνεται αναφορά σε προτάσεις για μελλοντικές προσεγγίσεις του συγκεκριμένου προβλήματος και βελτιώσεις στο συγκεκριμένο αλγόριθμο.

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF MECHANICAL & INDUSTRIAL ENGINEERING

Postgraduate Work

TRAIN DRIVER ROSTERING WITH LINEAR INTEGER PROGRAMMING

by

SKIADOPOULOS EVANGELOS

MECHANICAL ENGINEERING, ARISTOTLE UNIVERSITY OF THESSALONIKI, 2009

Submitted in partial fulfillment

requirements for

Postgraduate Specialization Diploma

2013

The approval of this postgraduate work by the Department of Mechanical and Industrial Engineering of the School of Engineering of the University of Thessaly does not imply acceptance of the writer's opinions (Law 5343/32 article 202 par.2).

Approved by:

First Examiner (Supervisor)
Dr. Athanasios Ziliaskopoulos
Professor, Department of Mechanical Engineering
University of Thessaly

Second Examiner
Dr. Dimitrios Pantelis
Professor, Department of Mechanical Engineering
University of Thessaly

Third Examiner
Dr. George Saharidis
Lecturer, Department of Mechanical Engineering,
University of Thessaly

Acknowledgements

This thesis is the end of my studies in obtaining my Master degree in MSc "**State-of-the-Art Design and Analysis Methods in Industry",** Industrial Management, so I would like to thank all those people who made my studies and this thesis possible.

At this moment of accomplishment, first of all, I would like to thank Dr. Ziliaskopoulos Athanasios for his trust on that thesis and his support. I would really like to thank Dr. Athanasios Lois for his valuable help and guidance throughout this work.

I am also grateful to the other members of the examining committee of my postgraduate work, Dr. Dimitrios Pantelis, Dr George Saharidis, for the close examination of my work and for the valuable knowledge I was given during my post-graduate studies.

I am thankful to my colleagues and friends Antonis Beslemes, Charalampos Papaeustathiou and Dimitrios Diamantis for the help and support during my studies.

Finally, I take this opportunity to express the profound gratitude from my deep heart to my beloved parents, Konstantinos and Paraskeui for their love and their continuous support, which I dedicate this postgraduate work.

Skiadopoulos Evangelos

.

.

TRAIN DRIVER ROSTERING WITH LINEAR INTEGER PROGRAMMING

by

SKIADOPOULOS EVANGELOS

MECHANICAL ENGINEERING, ARISTOTLE UNIVERSITY OF THESSALONIKI, 2009

Supervising Professor: Dr. Ziliaskopoulos Athanasios, Professor in Optimization Methods of Production/ Service Systems

Abstract

This study presents the experience of the approach in solving driver-rostering problem for a depot at Greek Railway Administration (Trainose). We built a brand new optimization algorithm and we managed to create a balanced timetable of shifts, according to labor rules and Greek Railway Administration's policies by using integer programming techniques.

At the beginning we gave a fully detailed problem description, discussing all the difficulties of that approach. Next we made a literature review of existing computerised systems. We also developed the theoretical background required for the solution of such problems, with a brief description of train driver's crew scheduling.

In the following chapters, we presented our model, describing the objective of that model and its constraints. The code that was used for the implementation of the algorithm was

written in C++ programming language, using IBM ILOG software to obtain our results. The necessary computational experiments were performed on one of the university's servers.

Finally we presented our results by using some mathematical examples and we summarized our conclusions and suggestions for future problem improvements.

Contents

List of Tables

List of Figures

List of Graphs

Chapter 1 INTRODUCTION

1.1 Introduction

Trainose S.A was established in December of 2005. Its main goal is to provide railway passenger and freight transport services in Greece. Trainose is trying to supply high quality transport services with social sensitiveness and respect for the Greek citizens.

With over 300 daily itineraries, the company's trains cover a railway network of more than 1.500 kilometers. TRAINOSE S.A. annually carries 15 million passengers and 4.5 million tons of freight. The main goal of the company is to deal with operational issues and make the railway transport cheaper and financially sustainable.

We focus on a crew rostering problem that Greek Railway Administration (Trainose) is facing. Crew rostering is a very common problem in Operations Research. The objective is to find an assignment of the crew to cover a planned time table such that we minimize cost, yet satisfy all constraints. Even though that is an easy to understand problem, creating an applicable timetable is proving to be quite complex, due to many operational rules and companies' policies.

In this thesis we present an integer linear programming algorithm in crew rostering while our main objective is to balance the working hours and days among the crew as evenly as possible and create an applicable time table of duties.

1.2 Structure of Postgraduate work

The remaining of this postgraduate work is structured as follows:

Chapter 2 reviews related works that have been published in the past about Crew Rostering and Crew Scheduling.

Chapter 3 presents the description of our problem. We perform a detailed analysis of its parameters and we have a close investigation of its difficulties.

Chapter 4 introduces the mathematical model that we develop in this thesis and we elaborate on its objective and constraints.

Chapter 5 provides some insight into our problem and we develop a methodology that can be used for its solution.

Chapter 6 generates the computational results of 4 numerical examples.

Chapter 7 provides the computational results of a real case in a Trainose's depot.

Chapter 8 presents the conclusions that we derived from the analysis of our results and points to some promising directions for future research.

Appendix A contains the C++ Programming Language source

Chapter 2 PROBLEM DESCRIPTION

In 1990's railway administrations started to investigate more thoroughly those issues by decomposing the main problem into two sub problems, crew rostering and crew scheduling.

Crew scheduling is actually the construction of a feasible set of duties, covering all trips in a given day. Crew rostering on the other hand is the creation of a cycle schedule, taking into consideration the labor legislation and many operational rules, which will be further discussed in the following paragraphs.

A main objective of crew management is the minimization of the number of train drivers needed to cover all feasible duties, generated in phase one (crew scheduling). This problem's constraint is rather weak, because the number of train drivers had already been determined by Trainose, so our main objective now is to balance the working hours and days among the crew as evenly as possible.

We have a lot of reasons to decompose that problem. First we have to generate feasible duties that start and end in the same depot. So this constrain imposes that each crew, within L days given by railway administration's policy, must return to its home depot and overlap very few consecutive days. In this problem we consider that a generated roster cannot include duties associated with different crew home locations.

Depots are not only the places where the trains may be loaded and unloaded, but also the places where a feasible duty starts and ends. They are located in big cities or towns, where crew is usually staying, resting and definitely changing shifts. If a member of the crew overnights at a depot, different from its home depot then this is called barracking.

A trip is starting and ending from one depot to another and it is characterized by its start and end time and its origin and destination depot. Roster is a sequence of trips. In the most common approaches it is a separate problem and deals with the generation of feasible duties, according to industrial and business regulation and must be covered in specific planning horizon.

Shift is the time period, during which a train driver is working. Train drivers will generally perform a sequence of individual train journeys with intermediate barracking stops before returning to their home base where they have a longer break before their next journey. Crew rostering in Railway application is originally difficult, because of the many operational constraints and a lot of possible feasible solutions. Adding constraints to a problem will usually make the space of feasible solutions smaller. However, algorithms that rely on iterative improvement, and in particular local search algorithms, might find it harder to generate feasible solutions in the neighborhood of the current solution.

Getting started with the generation of feasible duties we first have to know all the operational constraints and sequencing rules. The most important constraint is that we have to take care that every train driver cannot be assigned to cover more than a shift in a same day. Secondly, we have to take care that every shift is assigned to only one driver in a specific day, alternatively if the shift demands 2 train drivers, then those drivers are considered as a single unit.

A complete day is called a time interval of 24 hours starting at 00:00. A simple rest is called a day where no duty is performed. For each roster the number of double rests must be at least 40% of the total number of rest. So in 30 consecutive days, a single crew must have no more than 7 consecutive duties and his total working hours cannot exceed 170 hours.

As for the sequencing rules, the break between the end of a duty and the start of the duty in the following day lasts at least 18 hours. If both duties are overnight and one of them is a heavy overnight duty, then the break must last 22 hours. Naturally, in case of two consecutive heavy overnight duties, the break between them lasts at least 24 hours. Taking into consideration the labour rules the maximum working time per week is 45 hours.

Chapter 3 LITERATURE REVIEW

In this chapter, we will see some works that have been published in the past, which refer to various types of problems related to the crew scheduling and crew rostering problem. The assignment of work to individual crew members is a complex task for each public transport company. Traditionally, this process is split into two steps. In the first step, duties are constructed where a duty is the work for one crew member on a single day. These duties have to fulfill a lot of requirements. For instance, there is an upper bound on the length of each duty and there should be a break in each duty. This process is called crew scheduling. In the next step, rosters are created where sequences of duties are constructed. These sequences are assigned to the individual crew members. This problem is called crew rostering.

Crew scheduling is a very well-known problem which has been historically associated with airlines and mass transit companies; recently also railway applications have come on the scene. This now happens especially in Europe, where deregulation and privatization issues are forcing a re-organization of the rail industry and better productivity and efficient services are strongly required by the market and the public ownership. Therefore, this sector is showing an increasing interest in Operation Research and Management Science. Railway crew planning represents a hard problem due to both the dimensions and the operational constraints involved.

Considerable research has been carried out in scheduling public transport drivers since the late 1960's, limited in Bus and light railway applications. Train crew management involves the development of a duty timetable for each of the drivers (crew) to cover a given train timetable in a rail transport organization. This duty timetable is spread over a certain period, known as the roster planning horizon. Train crew management may arise either from the planning stage, when the total number of crew and crew distributions are to be determined, or from the operating stage when the number of crew at each depot is known as input data.

A natural formulation of Crew Scheduling problem, in terms of graphs, connects a node with each train station and directed arcs which indicate the planned train trip from one station to another. More specifically, one can define a directed graph G (V,A) having a note $J \in V$ for each train station and an arc $(i, j) \in A$ if and only if train station j can appear after train station i according to planned trips time table. So at this point the goal is the minimization of the feasible paths of G covering each node once.

There are two alternatives to model that problem in terms of integer linear programming. Let $\delta^{+}(\upsilon)$ and $\delta^{-}(\upsilon)$ represent the set of arcs that entering and leaving a specific train station $\upsilon \in V$. So we are creating a decision binary variable x_{ij} with each arc $(i, j) \in A$, where $x_{ij} = 1$ if we use arc $(i, j) \in A$ for our optimal solution and $x_{ij} = 0$ otherwise. We create a cost parameter C_{ij} for each arc $(i, j) \in A$.

$$\min \sum_{(i,j)\in A} C_{ij} x_{ij} \quad (1)$$

$$\sum_{(i,j)\in\delta^{+}(\upsilon)} x_{ij} = \sum_{(i,j)\in\delta^{-}(\upsilon)} x_{ij} = 1 \quad \upsilon \in V / D \quad (2)$$

$$\sum_{(i,j)\in\delta^{+}(\upsilon)} x_{ij} = \sum_{(i,j)\in\delta^{-}(\upsilon)} x_{ij} \quad \upsilon \in D \quad (3)$$

$$\sum_{(i,j)\in P} x_{ij} \le |P| - 1 \quad P \in \mathrm{P} \quad (4)$$

$$x_{ij} \in \{0,1\}, \quad (i,j) \in A \quad (5)$$

Where family P is a subset of P which cannot be part of any feasible solution.

Constraints (2)-(3) keep the balance of incomes and outcomes arcs in a specific node and each node $\upsilon \in V$ can be covered only once. Constraint (4) is a typical crew base constraint. At this point we determine a set of arcs that cannot be chosen because of operational constraints.

The problem with this model is that the cost of the optimum solution can be expressed as the sum of the cost associated with the arcs. We cannot use this model, when the cost of a circuit depends on the overall nodes (station) sequence or when we want to introduce cost per day parameter (e.g extra cost on Weekends). Second, we have to be very careful with the operational constraint (4), if it is so tight, then the linear programming relaxation of the model can be very week.

A variant of this model has a binary variable x_{ij}^{k} associated with arc $(i,j) \in A$ but also contains an index k assigned with crew type. So the cost parameter C_{ij}^{k} is the cost of

$(i, j) \in A$ performed by a crew of type k, where $C_{ij}^{k} = +\infty$ when the arc $(i, j) \in A$ is assigned to inappropriate crew type k, let K be the set of crew types. So:

$$\min \sum_{k \in K} \sum_{(i,j) \in A} C_{ij}^{k} x_{ij}^{k} \quad (6)$$

$$\sum_{(i,j) \in \delta^{+}(\upsilon)} x_{ij}^{k} = \sum_{(i,j) \in \delta^{-}(\upsilon)} x_{ij}^{k} \quad \upsilon \in V, k \in K \quad (7)$$

$$\sum_{(i,j) \in P} x_{ij}^{k} \leq |P| - 1 \quad P \in \mathrm{P}^{k}, k \in K \quad (8)$$

$$\sum_{k \in K} \sum_{(i,j) \in \delta^{+}(\upsilon)} x_{ij}^{k} = 1 \quad \upsilon \in V / D, (9)$$

$$x_{ij}^{k} \in \{0,1\}, \quad (i, j) \in A, k \in K \quad (10)$$

The only difference is that P^{k} is a subset of P and it cannot be part of any feasible solution for type k crews. Constraint (8) leads to tighter linear programming relaxation when the crew number is predetermined by the railway administrator. An obvious problem of this model is the increased number of variables and constraints.

Another approach to this problem is, let $C=\{C_1,...,C_n\}$ donate the collection of the simple circuits of G corresponding to a feasible duty for a crew. We assign a cost c_j to every circuit C_j that covers the node set I_j. So we create a binary decision variable $y_j=1$ if C_j is part of the optimal solution, and 0 otherwise.

$$\min \sum_{(i,j)\in A} c_j y_j \quad (11)$$

Subject to

$$\sum_{j:\upsilon\in I_j} y_j = 1 \quad \upsilon \in V \setminus D \quad (12)$$

$$\sum_{j\in S} y_j \le |S| - 1 \quad s \in S \quad (13)$$

$$y_j \in \{0,1\}, \quad j = 1,...,n \quad (14)$$

British Approach

In the early 90's, British Rail had a very interested approach to this problem, using interactive techniques. They decomposed their problem into three types of schedules: Long distance, local services and ancillary work. For long distance schedules, British Rail tried to specify certain relief points, in order to retrieve the timetables of trains between these points and derive a solution using an assignment technique.

For the ancillary work schedules, conventional batch processing approaches were used. The ancillary work pieces were matched with the spare capacity of the driver's schedule and in case of infeasibility; extra drivers were logged in the system.

The local service schedule was by far more complex. In this case, trains work in much shorter distances within a small area. So they approached this problem using color graphics to represent trains on a VDU. Shifts were built up, one at a time and they were checked by the system. If the work was covered, it was removed from the display.

An attempt had been made to use mathematical programming methods by British Rail for the train driver scheduling problem, but this approach was later abandoned. By the time when British Rail was privatised in the early 1990's, there was still no known automatic driver scheduling system being used by British Rail.

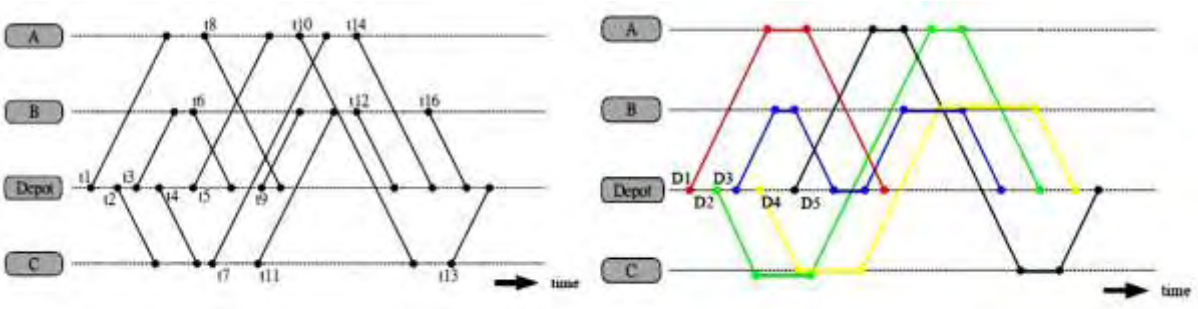


Figure 3-1: Generation of feasible duties that cover all trips

The New Jersey Railway Approach

The scheduling system that was developed in 1980 was to enable the management in order to analyze quickly the implication of work rule changes on labour costs for union negotiations. There was no mention of the size of the problem or the number of crew depots involved. This approach helped to the introduction of the "deadheading trips" well known as "drivers travel as passengers". Similar to route knowledge restriction, crew members could only be assigned to a single line or group of lines according to the crew's depots.

The problem was solved in three steps. First, by solving a set covering problem using a mathematical programming method, they managed to generate feasible duties that cover all vehicle work (trips). Next they created cost parameters per duty and then, using mathematical programming approaches they determined the schedule.

The fact that there is a high percentage of deadhead loops suggests that the shift generation process seems to be very restrictive and very few alternatives are available for the mathematical programming stage.

The Italian State Railways Approach

The Italian State Railways operates a vast network with a crew workforce of about 25,000 drivers and 15,000 conductors located in about 50 depots. The system described is a planning system for locomotive scheduling, crew scheduling and rostering. For crew scheduling, a traditional approach was used, which involves two phases based on a set covering model.

The first phase focuses on generating a very large number of feasible shifts while the second phase selects the best subset of all the generated shifts to cover all the 'trips' at minimum cost. The shift generation phase sequences a number of ‘trips’ to form a shift which must start and end at the same depot. A ‘trip’ is defined as segments of train journey which must be serviced by the same crew without rest. However, it is not mentioned whether there are any more relief opportunities on each trip.

The simplifications on the problem would have prevented some crucial shifts from being generated. It is likely that the shifts generated are very restrictive in order to reduce the possible number of combinations and as a result the schedules produced would not be anywhere near the optimum, if some of the crucial shifts were missing.

Chapter 4 MODEL DEVELOPMENT

In this section, we present the integer programming model that was developed for the problem under consideration. We use the following mathematical notation:

Index:

i: Is the index for train drivers, every driver is assigned with a number, driver's ID and $i \in I$. I is the group of the Drivers from a specific depot

j: Is the index for feasible duties, generated in phase one (Crew scheduling) and $j \in J$. J is the set of feasible duties, that are starting and ending in the same depot.

d: Is the index for days, $d \in D$. D is the time horizon for the creation of an applicable time table. Train drivers are assigned in a cyclic schedule.

m: Is an artificial index for days, $m \in M$. We use it to create subsets for 2 consecutive days

Decision Variable:

D_{ijd}: Is a binary decision variable and indicates if the i train driver is assigned to j duty at d day with $i \in I$, $j \in J$, $d \in D$.

$$D_{ijd} = \begin{cases} 1, & if \quad driver \quad i \quad assigned \quad to \quad duty \quad j \quad on \quad day \quad d \\ 0, & if \quad driver \quad i \quad has \quad not \quad assigned \quad to \quad duty \quad j \quad on \quad day \quad d \end{cases}$$

Parameters:

I : It is the group of the train drivers of a depot

J : It is the set of feasible duties of a depot

D : It is the set of days - our planning horizon

WHW : Is the maximum working hours per week for every train driver taking in consideration the weights for overtime, working on Weekends, medium and late duties

T_d : It is a cost parameter per day d .

K_j : It is a cost parameter per duty j .

L_j : It is the set of late duties. Late duties start from 22.00 – 05.59

E_j : It is the set of early duties. Early duties start from 06.00 – 13.59

M_j : It is the set of medium duties. Medium duties start from 14.00 – 21.59

KL_j : It is the set of the extra cost per late duty j

KM_j : It is the set of the extra cost per medium duty j

Then, the referenced problem can be formulated as follows:

$$\min z = \sum_i \sum_j \sum_d D_{ijd} * T_d * K_j * KL_j * KM_j \qquad j \in J, i \in I, d \in D \qquad (1)$$

Code Implementation:

```
IloExpr expr1(env);
        for (i=0;i<imax;i++){
                for (j=0;j<jmax;j++){
                        for (d=0;d<dmax;d++){
                        expr1+=Dijd[i][j][d]*T[d]*KL[j]*K[j]*KM[j];
                        }
                }
        }

model.add(IloMinimize(env, expr1));
expr1.end();
```

The objective function (1) minimizes the global cost of all drivers. The first summation is associated with the total number of train drivers from a specific depot. The second summation is associated with the set of feasible duties that start and end in the same depot and the third summation is associated with the days of our planning horizon.

Constraints

$$\sum_i D_{ijd} = 1 \quad \forall j,d \quad j \in J, i \in I, d \in D \qquad (2)$$

Code Implementation:

```
//--------------------(2) Set of constraints: Only one Duty per Driver -----------

IloRangeMatrix2x2 SumDjd(env,0);
        for (j=0;j<jmax;j++){
                IloRangeArray SumDd(env,0);
                        for (d=0;d<dmax;d++){
                                IloExpr expr(env,0);
                                        for (i=0;i<imax;i++){
                                                expr+=Dijd[i][j][d];
                                                }
                                char Ipir_MIX[60];
                                sprintf(Ipir_MIX,"SumDijd(j%d,d%d)",j,d);
                                float LB=1,UB=1;
                                IloRange SumD(env,LB,expr,UB,Ipir_MIX);
                                model.add(SumD);
                                SumDd.add(SumD);
                                expr.end();
                                }
                        SumDjd.add(SumDd);
                        }
```

The first set of constraints (2) is used to make sure that every driver i, every day d is assigned to one and only one feasible duty.

$$\sum_{j} D_{ijd} \leq 1 \quad \forall i,d \quad j \in J, i \in I, d \in D \qquad (3)$$

Code Implementation:

```
//------------ (2) Set of constraints: Only one Driver per Duty------------------------------

IloRangeMatrix2x2 Sum2Did(env,0);
        for (i=0;i<imax;i++){
                IloRangeArray Sum2Dd(env,0);
                        for (d=0;d<dmax;d++){
                                IloExpr expr(env,0);
                                        for (j=0;j<jmax;j++){
                                        expr+=Dijd[i][j][d];
                                        }
                                char Ipir_MIX2[60];
                                sprintf(Ipir_MIX2,"Sum2Dijd(j%d,d%d)",j,d);
                                float LB=-IloInfinity,UB=1;
                                IloRange Sum2D(env,LB,expr,UB,Ipir_MIX2);
                                model.add(Sum2D);
                                Sum2Dd.add(Sum2D);
                                expr.end();
                                }
                        Sum2Did.add(Sum2Dd);
                        }
```

The second set of constraints (3) is used to forbid a duty j, every day d, to be served by more than one train drivers. At this point we consider that every duty is covered by only one driver. If a duty needs to be covered by more than one driver, we generate an extra driver at the end of our feasible timetable.

$$\sum_{j}\sum_{d} D_{ijd} * T_d * K_j * KL_j * KM_j \le WHW \quad \forall i \quad j \in J, i \in I, d \in D \qquad (4)$$

Code Implementation:

```
//-------------------------------------------------------------------------
//-----------(3) Set of constraints: Maximum WHW for every driver----------


        IloRangeArray Sum3Di(env,0);
                for (i=0;i<imax;i++){
                        IloExpr expr(env,0);
                        for (j=0;j<jmax;j++){
                                for (d=0;d<dmax;d++){
                                        expr+=Dijd[i][j][d]*T[d]*KL[j]*K[j]*KM[j];
                                        }
                                }
                                char Ipir_MIX3[60];
                                sprintf(Ipir_MIX3,"Sum3Dijd(j%d,d%d)",j,d);
                                float LB=-IloInfinity,UB=77;
                                IloRange Sum3D(env,LB,expr,UB,Ipir_MIX3);
                                model.add(Sum3D);
                                Sum3Di.add(Sum3D);
                                expr.end();
                        }
```

Constraint set (4) is determining the upper bound of working hours per week, for every train driver i. We will describe this parameter in more detail in the following paragraphs.

$$\sum_{j}\sum_{d=m}^{m+1} D_{ijd} * E_j + D_{ijd+1} * E_j \leq 1 \quad \forall i \quad j \in J, i \in I, d \in D, m \in M \quad (5)$$

Code Implementation:

```
//-----------(4) Set of constraints: No early duty, after early duty----------

        IloRangeMatrix2x2 Sum4Dmi (env,0);
                for (m=0;m<mmax;m++){
                        IloRangeArray Sum4Di(env,0);
                                for (i=0;i<imax;i++){
                                        IloExpr expr(env,0);
                                                for (j=0;j<jmax;j++){
                                                        for (d=m;d<m+1;d++){
                                        expr+=Dijd[i][j][d]*E[j]+Dijd[i][j][d+1]*E[j];
                                        }
                                }

                                char Ipir_MIX4[60];
                                sprintf(Ipir_MIX4,"Sum4Dijd(i%d)",i);
                                float LB=-IloInfinity,UB=1;
                                IloRange Sum4D(env,LB,expr,UB,Ipir_MIX4);
                                model.add(Sum4D);
                                Sum4Di.add(Sum4D);
                                expr.end();
                                }
                        Sum4Dmi.add(Sum4Di);
                        }
```

Constraint set (5) ensures that for every two consecutive days, a driver cannot have an early duty, after an early duty.

$$\sum_{j}\sum_{d=m}^{m+1} D_{ijd} * M_j + D_{ijd+1} * M_j + D_{ijd+1} * E_j \leq 1 \quad \forall i \quad j \in J, i \in I, d \in D, m \in M \quad (6)$$

Code Implementation:

```
//-------------------------------------------------------------------------
//-----------(5) Set of constraints: No early or medium duty, after medium duty----------

        IloRangeMatrix2x2 Sum5Dmi (env,0);
                for (m=0;m<mmax;m++){
                        IloRangeArray Sum5Di(env,0);
                                for (i=0;i<imax;i++){
                                        IloExpr expr(env,0);
                                                for (j=0;j<jmax;j++){
                                                        for (d=m;d<m+1;d++){
        expr+=Dijd[i][j][d]*M[j]+Dijd[i][j][d+1]*M[j]+Dijd[i][j][d+1]*E[j];
                                                        }
                                        }

                                        char Ipir_MIX5[60];
                                        sprintf(Ipir_MIX5,"Sum5Dijd(i%d)",i);
                                        float LB=-IloInfinity,UB=1;
                                        IloRange Sum5D(env,LB,expr,UB,Ipir_MIX5);
                                        model.add(Sum5D);
                                        Sum5Di.add(Sum5D);
                                        expr.end();
                                        }
                                Sum5Dmi.add(Sum5Di);
                                }
```

Constraint set (6) ensures that for every two consecutive days, a driver cannot have an early or a medium duty, after a medium duty.

$$\sum_{j}\sum_{d=m}^{m+1} D_{ijd} * L_j + D_{ijd+1} * L_j + D_{ijd+1} * M_j + D_{ijd+1} * E_j \leq 1 \quad \forall i \quad j \in J, i \in I, d \in D, m \in M \quad (7)$$

Code Implementation:

```
//-------------------------------------------------------------------------
//-----------(6) Set of constraints: Day off after late duty-------------


        IloRangeMatrix2x2 Sum6Dmi (env,0);
                for (m=0;m<mmax;m++){
                        IloRangeArray Sum6Di(env,0);
                                for (i=0;i<imax;i++){
                                        IloExpr expr(env,0);
                                                for (j=0;j<jmax;j++){
                                                        for (d=m;d<m+1;d++){

expr+=Dijd[i][j][d]*L[j]+Dijd[i][j][d+1]*L[j]+Dijd[i][j][d+1]*M[j]+Dijd[i][j][d+1]*E[j
];
                                                        }
                                        }

                                        char Ipir_MIX6[60];
                                        sprintf(Ipir_MIX6,"Sum6Dijd(i%d)",i);
                                        float LB=-IloInfinity,UB=1;
                                        IloRange Sum6D(env,LB,expr,UB,Ipir_MIX6);
                                        model.add(Sum6D);
                                        Sum6Di.add(Sum6D);
                                        expr.end();
                                        }
                                Sum6Dmi.add(Sum6Di);
                        }
```

Constraint set (7) ensures that for every two consecutive days, a driver must have a day off, after a late duty.

$$\sum_j D_{ij7} * E_j + D_{ij1} * E_j \leq 1 \quad \forall i \quad j \in J, i \in I \quad (8)$$

Code Implementation:

```
//-------------------------------------------------------------------------
//-----------(7) Set of constraints: No early duty, after early duty for D=7 and D=0------

        IloRangeArray Sum7Di(env,0);
                for (i=0;i<imax;i++){
                        IloExpr expr(env,0);
                                for (j=0;j<jmax;j++){
                                        for (d=dmax-1;d<dmax;d++){
                                expr+=Dijd[i][j][dmax-1]*E[j]+Dijd[i][j][0]*E[j];
                                }
                        }

                        char Ipir_MIX7[60];
                        sprintf(Ipir_MIX7,"Sum7Dijd(i%d)",i);
                        float LB=-IloInfinity,UB=1;
                        IloRange Sum7D(env,LB,expr,UB,Ipir_MIX7);
                        model.add(Sum7D);
                        Sum7Di.add(Sum7D);
                        expr.end();
                        }
```

$$\sum_j D_{ij7} * M_j + D_{ij1} * M_j + D_{ij1} * E_j \leq 1 \quad \forall i \quad j \in J, i \in I \quad (9)$$

Code Implementation:

```
//-------------------------------------------------------------------------
//------(8) Set of constraints: No early or medium duty, after medium duty for D=7 and D=0---

        IloRangeArray Sum8Di(env,0);
                for (i=0;i<imax;i++){
                        IloExpr expr(env,0);
                                for (j=0;j<jmax;j++){
                                        for (d=dmax-1;d<dmax;d++){
        expr+=Dijd[i][j][dmax-1]*M[j]+Dijd[i][j][0]*M[j]+Dijd[i][j][0]*E[j];
                                        }
                                }

                                char Ipir_MIX8[60];
                                sprintf(Ipir_MIX8,"Sum8Dijd(i%d)",i);
                                float LB=-IloInfinity,UB=1;
                                IloRange Sum8D(env,LB,expr,UB,Ipir_MIX8);
                                model.add(Sum8D);
                                Sum8Di.add(Sum8D);
                                expr.end();
                                }
```

$$\sum_j D_{ij7} * L_j + D_{ij1} * L_j + D_{ij1} * M_j + D_{ij1} * E_j \leq 1 \quad \forall i \quad j \in J, i \in I \quad (10)$$

Code Implementation:

```
//-------------------------------------------------------------------------
//------(9) Set of constraints: Day off after Late duty for D=7 and D=0---


        IloRangeArray Sum9Di(env,0);
                for (i=0;i<imax;i++){
                        IloExpr expr(env,0);
                                for (j=0;j<jmax;j++){
                                        for (d=dmax-1;d<dmax;d++){

expr+=Dijd[i][j][dmax-1]*L[j]+Dijd[i][j][0]*L[j]+Dijd[i][j][0]*M[j]+Dijd[i][j][0]*E[j];
                                        }
                        }

                        char Ipir_MIX9[60];
                        sprintf(Ipir_MIX9,"Sum9Dijd(i%d)",i);
                        float LB=-IloInfinity,UB=1;
                        IloRange Sum9D(env,LB,expr,UB,Ipir_MIX9);
                        model.add(Sum9D);
                        Sum9Di.add(Sum9D);
                        expr.end();
                        }
```

Constraint sets (8) - (9) - (10) are crucial for the construction of a cyclic schedule. We consider, days d=7 and d=0 as consecutive.

We are going to describe our cost parameters in more detail. First we have the cost per day parameter for a single week.

Days	1	2	3	4	5	6	7
Cost	1	1	1	1	1	1.5	2

Table 4-1: Cost per day

Then we are going to describe cost parameters that are associated with feasible duties that generated in phase one. We assume that Medium duties (start at 14.00) have an extra cost. This cost is the product of working hours multiplied by 1.2 factor and Late duties (start at 22.00) have an extra cost, which is the product of working hours multiplied by 1.5 factor.

Now, we have to determine the input of the train drivers and **WHW** (working hours per week). Our goal is the extraction of empirical values that first reduce the number of train drivers needed in every case and then balance the working hours among crew.

Our solution approach is the following: We are using the same algorithm without (2) constraint set and we are using as an input imax=1. With this procedure, we calculate the minimum $WHW_{\min}$ of a driver. Then we create a maximization problem and we calculate the maximum $WHW_{\max}$. WHW parameter is very important for the balance of the working hours among the crews. We also take care of the computational time, which is very important for those applications.

$$\min/\max z = \sum_{1}\sum_{j}\sum_{d} D_{ijd} * T_d * K_j * KL_j * KM_j \qquad j \in J, i = 1, d \in D \quad (1)$$

s.t

$$\sum_{j} D_{ijd} \leq 1 \quad \forall i, d \qquad j \in J, i \in I, d \in D \qquad (3)$$

$$\sum_{j}\sum_{d} D_{ijd} \leq 5 \quad \forall i \qquad j \in J, i \in I, d \in D \qquad (4)$$

$$\sum_{j}\sum_{d=m}^{m+1} D_{ijd} * E_j + D_{ijd+1} * E_j \leq 1 \quad \forall i \qquad (5)$$

$$\sum_{j}\sum_{d=m}^{m+1} D_{ijd} * M_j + D_{ijd+1} * M_j + D_{ijd+1} * E_j \leq 1 \quad \forall i \qquad (6)$$

$$\sum_{j}\sum_{d=m}^{m+1} D_{ijd} * L_j + D_{ijd+1} * L_j + D_{ijd+1} * M_j + D_{ijd+1} * E_j \leq 1 \quad \forall i \qquad (7)$$

$$\sum_{j} D_{ij7} * E_j + D_{ij1} * E_j \leq 1 \quad \forall i \qquad (8)$$

$$\sum_{j} D_{ij7} * M_j + D_{ij1} * M_j + D_{ij1} * E \leq 1 \quad \forall i \qquad (9)$$

$$\sum_{j} D_{ij7} * L_j + D_{ij1} * L_j + D_{ij1} * M_j + D_{ij1} * E_j \leq 1 \quad \forall i \qquad (10)$$

Constraint set (4) does not allow drivers to have more than 5 duties per week.

Let's give an example to make this clear. We have a problem of 5 feasible duties and we are giving the cost parameters:

Duties	1	2	3	4	5
Type	E	L	M	M	E
Duration	8	8	10	9	8

Table 4-2: Example 0 Duty's type and duration

E: Early Duty

M: Medium Duty

L: Late duty

We know that weekends have an extra cost

Days	1	2	3	4	5	6	7
Cost	1	1	1	1	1	1.5	2

Table 4-3: Example 0 Cost per day

Late and medium duties have a weight 1.5 and 1.2 respectfully.

We use our maximization and minimization algorithm to create an upper and lower bound in WHW.

Min	5(E)	4(M)	2(L)		1 (E)	4(M)		**55**
Max		5(E)	3(M)	2(L)		3(M)	2(L)	**74**

Table 4-4: Example 0 WHWmin and WHWmax

We can balance the working hours among crews efficiently by using this formula

$$WHW = \frac{WHW_{\min} + WHW_{\max}}{2}.$$

Next we have to determine the number of drivers. We introduce another binary decision variable DI_i which:

$$DI_i = \begin{cases} 1, & if \quad we \quad use \quad driver \quad i \quad in \quad our \quad timetable \\ 0, & if \quad we \quad don't \quad use \quad driver \quad i \quad in \quad our \quad timetable \end{cases}$$

And we create a cost parameter IN_i for every extra driver to our schedule. So, by changing the objective function, we minimize the number of drivers needed in every case.

$$\min z = \sum_{1} \sum_{j} \sum_{d} D_{ijd} * T_d * K_j * KL_j * KM_j + DI_i * IN_i \qquad j \in J, i \in I, d \in D \qquad (1)$$

Using a random number of drivers as an input, the algorithm had to determine the minimum number of drivers needed to cover all shifts every day. However, we decided not to use this parameter and this decision variable, because it would increase the computational time needed. So by executing several numerical examples we managed to extract empirical experimental formula to determine the minimum number of drivers needed in any case.

So we found out that, the relation between train drivers and shifts for the creation of a feasible time table, is

$$No.Drivers \geq No.Shifts * 1.4$$

Chapter 5 SOLUTION METHODOLOGY

In this chapter, we develop our solution methodology. Our duties have 3 main characteristics:

1) **Origin Depot:** It is the depot where the train drivers start and end their duties. In our case all duties start and end in the same depot.
2) **Starting time:** By this time, train drivers must be in their position in a specific depot.
3) **Ending time:** By this time, train drivers must be back in their origin depot.

First we have to categorize our shifts according to their starting time. We have three main categories:

1) **Early Duties (E)** start between 06.00 and 13.59. This situation is considered as a "normal operation".
2) **Medium Duties (M)** start between 14.00 till 21.59. The working hours of this category are multiplied by a weight equal to 1.2.
3) **Late Duties (L)** that start from 22.00 till 05.59. The working hours of this category are multiplied by a weight equal to 1.5.

Next we have to determine the number of the train drivers using this formula.

$$No.Drivers \geq No.Shifts * 1.4$$

Then we calculate $WHW_{\min/\max}$ and we create an upper and lower bound for every train driver for his working hours per week.

We consider that a normal operation condition is 5 duties per week. So the algorithm is searching for the maximum working hours duties to assign them to the worst days of the week creating a maximum working hour time table for every driver.

By solving the exact opposite problem we create a minimum working hour time table for every driver. So in order to balance the working hours among the crew, we should choose the value of WHW between these bounds

$$WHW_{\min} \leq WHW \leq WHW_{\max}$$

Solving the algorithm, we create a cyclic week timetable according to our needs. Then we begin to generate extra train drivers for:

1) Covering duties that need more than one driver
2) Train drivers that ask days off for vacation
3) Train drivers that ask days off for medical reasons

Chapter 6 COMPUTATIONAL IMPLEMENTATION

6.1 Numerical example 1

Inputs

$I = 7$ Drivers needed to cover all duties

$J = 5$ Feasible duties to cover all trips

$D = 7$ Time horizon for the construction of a cyclic schedule

Parameter values:

Duties 1 and 5 are early duties that start between 06.00 and 13.59

Duties 3 and 4 are medium duties that start between 14.00 and 21.59

Duty 2 is late duty that starts between 22.00 and 05.59

Duties	**1**	**2**	**3**	**4**	**5**
Type	E	L	M	M	E
Duration	8	8	10	9	8
Cost	**8**	**12**	**12**	**10.8**	**8**

Table 6-1: Numerical Example 1 - Duty's type and duration

Cost per day

A duty on Saturday is multiplied by 1.5 and a duty on Sunday is multiplied by 2

Days	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Cost	1	1	1	1	1	1.5	2

Table 6-2: Numerical Example 1 – Cost per day

First, as an input we choose 7 drivers because, considering our solution algorithm, it is the minimum number of drivers for the construction of a feasible time table. Next, we calculate the WHWmin and WHWmax as follows. WHWmin is the minimum feasible working hours per week, it is actually the minimum combination between days and duties, according to labour rules for a single driver. WHWmax on the other hand is the maximum feasible working hours per week for a single driver.

Days	M	T	W	T	F	S	S	Total
WHW Min	5(E)	4(M)	2(L)		1 (E)	4(M)		
Min Cost	**8**	**9*1.2**	**8*1.5**		**8**	**9*1.2*1.5**		**55**
WHW Max		5(E)	3(M)	2(L)		3(M)	2(L)	
Max Cost		**8**	**10*1.2**	**8*1.5**		**10*1.2**	**8*1.5*2**	**74**

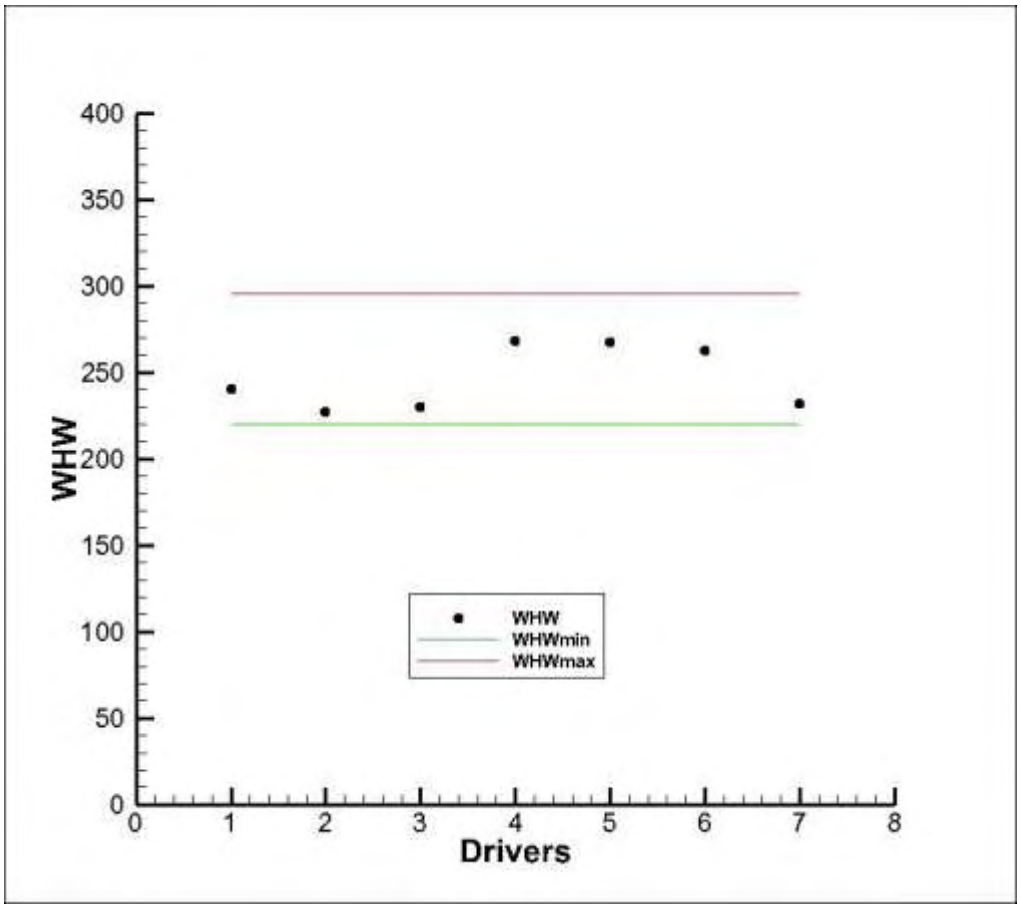
Table 6-3: Numerical Example 1 – WHWmin and WHWmax

$WHW = \frac{WHW_{\min} + WHW_{\max}}{2} = \frac{74+55}{2} = 64.5$ We create a feasible time table WHW=67

	Days							
Drivers	**1**	**2**	**3**	**4**	**5**	**6**	**7**	Σύνολο
1	3(M)		5(E)	3(M)	2(L)		1 (E)	60
2	1 (E)	4(M)	2(L)		5(E)	3(M)		56,8
3	4(M)	2(L)		1 (E)	4(M)		5(E)	57,6
4		1 (E)	4(M)		1 (E)	4(M)	2(L)	67
5	2(L)		1 (E)	4(M)		5(E)	3(M)	66,8
6		5(E)	3(M)	2(L)		1 (E)	4(M)	65,6
7	5(E)	3(M)		5(E)	3(M)	2(L)		58

Table 6-4: Numerical Example 1 – Train Driver's Rostering Timetable

We managed to create a feasible timetable with maximum variance between crew 17.96%. **CPU time=0.05 sec. (Intel® Core ™ i5-2430M CPU @ 2.4GHz – RAM 6,00 GB)**



Graph 6-1: Numerical Example 1 – Dispersion of WHW among crew

6.2 Numerical example 2

Inputs

$I = 10$ Drivers needed to cover all duties

$J = 7$ Feasible duties to cover all trips

$D = 7$ Time horizon for the construction of a cyclic schedule

Parameter values:

Duties 1, 4 and 7 are early duties that start between 06.00 and 13.59

Duties 3 and 5 are medium duties that start between 14.00 and 21.59

Duties 2 and 6 are late duty that start between 22.00 and 05.59

Duties	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Type	E	L	M	E	M	L	E
Duration	8	8	10	9	8	8	10
Cost	8	12	12	9	9.6	12	10

Table 6-5: Numerical Example 2 - Duty's type and duration

Cost per day

A duty on Saturday, is multiplied by 1.5 and a duty on Sunday is multiplied by 2

Days	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Cost	1	1	1	1	1	1.5	2

Table 6-6: Numerical Example 2 – Cost per day

First, as an input we choose 10 drivers. Considering our solution algorithm, it is the minimum number of drivers for the construction of a feasible time table. Next we calculate the WHWmin and WHWmax as follows. WHWmin is the minimum feasible working hours per week, it is actually the minimum combination of days and duties, according to labour rules for a single driver. WHWmax on the other hand is the maximum feasible working hours per week for a single driver.

Days	M	T	W	T	F	S	S	Total
WHW Max		7(E)	3(M)	6(L)		3(M)	2(L)	
Max Cost		**10**	**12**	**12**		**12*1.5**	**12*2**	**76**
WHW Min	1(E)	5(M)	2(L)		1(E)	5(M)		
Min Cost	**8**	**9.6**	**12**		**8**	**9.6*1.5**		**52**

Table 6-7: Numerical Example 2 – WHWmin and WHWmax

$$WHW = \frac{WHW_{\min} + WHW_{\max}}{2} = \frac{76+52}{2} = 64$$

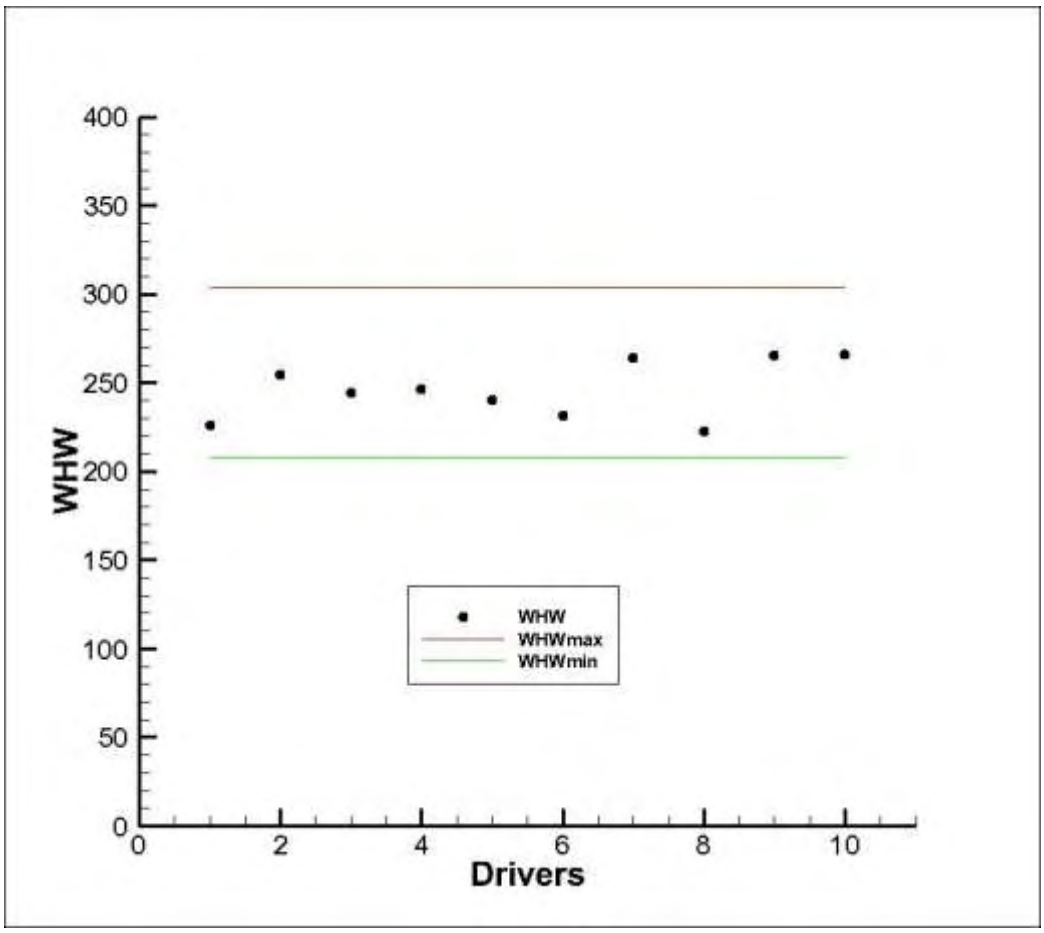
We create a feasible time table WHW=67

	Days							
Drivers	**1**	**2**	**3**	**4**	**5**	**6**	**7**	Σύνολο
1	1(E)	6(L)		4(E)	5(M)	6(L)		56,6
2	3(M)		7(E)	5(M)	6(L)		7(E)	63,6
3	2(L)		4(E)	3(M)	2(L)		1(E)	61
4	5(M)	2(L)		7(E)	3(M)		4(E)	61,6
5	7(E)	3(M)	6(L)		7(E)	3(M)		60
6	4(E)	5(M)	2(L)		4(E)	2(L)		57,84
7	6(L)		1(E)	6(L)		7(E)	5(M)	66,2
8		7(E)		1(E)		4(E)	2(L)	55,5
9		1(E)	3(M)		1(E)	5(M)	6(L)	66,4
10		4(E)	5(M)	2(L)		1(E)	3(M)	66,6

Table 6-7: Numerical Example 2 – Train Driver's Rostering Timetable

We managed to create a feasible timetable with 20% maximum variance between crew

CPU time= 0.58 sec. (Intel® Core ™ i5-2430M CPU @ 2.4GHz – RAM 6,00 GB)



Graph 6-2: Numerical Example 2 – Dispersion of WHW among crew

6.3 Numerical example 3

Inputs

$I = 21$ Drivers needed to cover all duties

$J = 15$ Feasible duties to cover all trips

$D = 7$ Time horizon for the construction of a cyclic schedule

Parameter values:

Duties 1, 4, 7, 10, 11 and 15 are early duties that start between 06.00 and 13.59

Duties 2, 3, 5, 9 and 14 are medium duties that start between 14.00 and 21.59

Duties 6, 8, 12 and 13 are late duty that start between 22.00 and 05.59

Duties	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Type	E	M	M	E	M	L	E	L
Duration	8	8	10	9	8	8	10	10
Duties	**9**	**10**	**11**	**12**	**13**	**14**	**15**	
Type	M	E	E	L	L	M	E	
Duration	9	8	9	8	9,5	8,5	8	

Table 6-8: Numerical Example 3 - Duty's type and duration

Cost per day

A duty on Saturday is multiplied by 1.5 and a duty on Sunday is multiplied by 2

Days	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Cost	1	1	1	1	1	1.5	2

Table 6-9: Numerical Example 3 – Cost per day

First, as an input we choose 21 drivers. Considering our solution algorithm, it is the minimum number of drivers for the construction of a feasible time table. Next we calculate the WHWmin and WHWmax as follows. WHWmin is the minimum feasible working hours per week, it is actually the minimum combination of days and duties, according to labour rules for a single driver. WHWmax on the other hand is the maximum feasible working hours per week for a single driver.

Days	M	T	W	T	F	S	S	
WHWmax		7(E)	3(M)	8(L)		3(M)	8(L)	
Max Cost		**10**	**12**	**15**		**12*1.5**	**15*2**	**85**
WHWmin	15(E)	5(M)	6(L)		1(E)	2(M)		
Min Cost	**8**	**9.6**	**12**		**8**	**9.6*1.5**		**52**

Table 6-10: Numerical Example 3 – WHWmin and WHWmax

$$WHW = \frac{WHW_{\min} + WHW_{\max}}{2} = \frac{85+52}{2} = 68.5$$

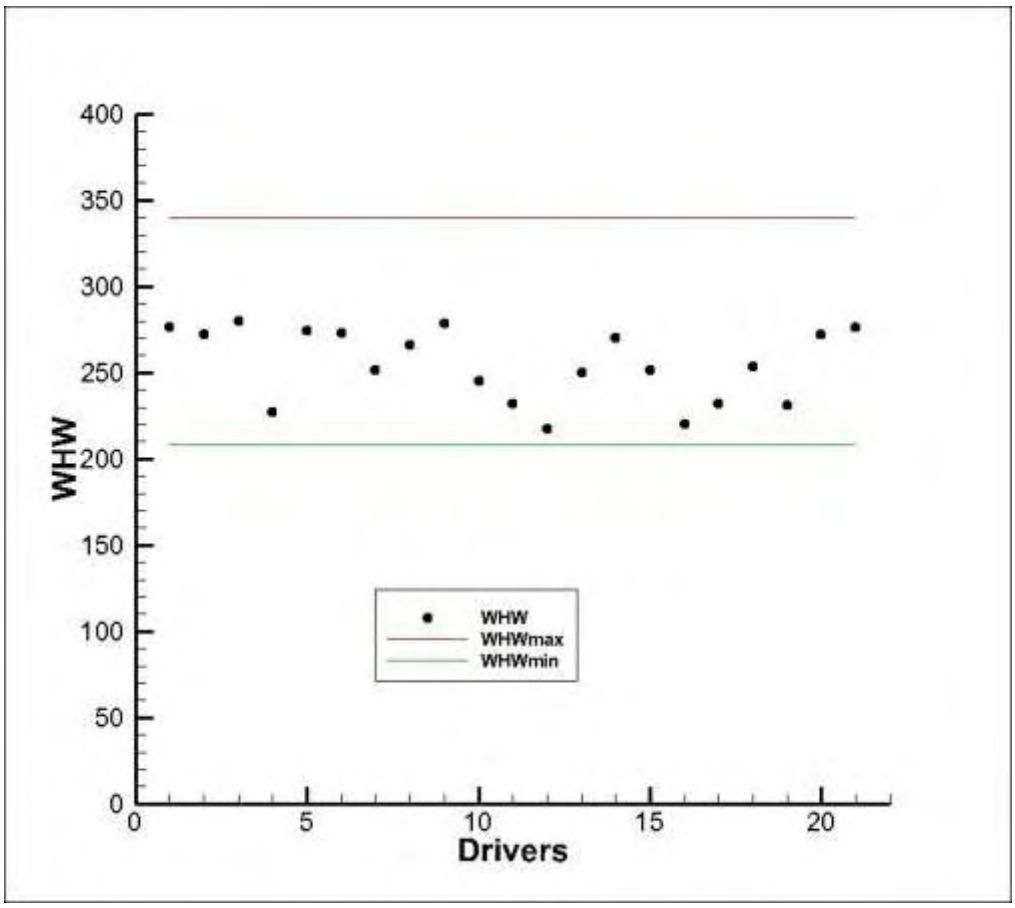
We create a feasible time table WHW=70

	Days							
Drivers	**1**	**2**	**3**	**4**	**5**	**6**	**7**	Σύνολο
1	12(L)		7(E)	9(M)		4(E)	14(M)	69,05
2	6(L)		1(E)	14(M)		1(E)	3(M)	68,1
3		1(E)	2(M)	6(L)		15(E)	8(L)	70
4	14(M)	13(L)		10(E)	6(L)		4(E)	56,8
5	15(E)	14(M)		1(E)	3(M)	6(L)		68,5
6	2(M)		4(E)	5(M)	8(L)		11(E)	68,3
7	11(E)	6(L)		11(E)	14(M)	12(L)		62,8
8	10(E)	3(M)	13(L)		15(E)	3(M)		66,6
9		4(E)	9(M)	8(L)		10(E)	5(M)	69,65
10	1(E)	9(M)	8(L)		4(E)	13(L)		61,25
11	9(M)		15(E)	2(M)	13(L)		1(E)	58,2
12	3(M)	12(L)		4(E)	2(M)		7(E)	54,4
13		15(E)	5(M)		10(E)	2(M)	13(L)	62,6
14	4(E)	5(M)		15(E)	5(M)	8(L)		67,6
15		10(E)	14(M)	12(L)		7(E)	9(M)	62,825
16	13(L)		10(E)	13(L)		11(E)	2(M)	55,2
17	5(M)	8(L)		7(E)	9(M)		10(E)	58,05
18		7(E)	3(M)		1(E)	5(M)	6(L)	63,3
19		11(E)	12(L)		11(E)	14(M)	12(L)	57,8
20	7(E)	2(M)	6(L)		7(E)	9(M)		68,2
21	8(L)		11(E)	3(M)	12(L)		15(E)	69,1

Table 6-11: Numerical Example 3 – Train Driver's Rostering Timetable

We managed to create a feasible timetable with 22.28% maximum variance between crew

CPU time= 3.52 sec. (Intel® Core ™ i5-2430M CPU @ 2.4GHz – RAM 6,00 GB)



Graph 6-3: Numerical Example 3 – Dispersion of WHW among crew

6.4 Numerical example 4

Inputs

$I = 43$ Drivers needed to cover all duties

$J = 30$ Feasible duties to cover all trips

$D = 7$ Time horizon for the construction of a cyclic schedule

Parameter values:

Duties (E) are early duties that start between 06.00 and 13.59

Duties (M) are medium duties that start between 14.00 and 21.59

Duties (L) are late duty start that between 22.00 and 05.59

Duties	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Type	E	L	M	E	M	L	E	M	L	E
Duration	8	8	10	9	8	8	10	10	9	8
Duties	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Type	E	M	E	M	L	L	M	E	M	E
Duration	**9**	**8**	**9,5**	**8,5**	**8**	**9**	**8**	**9**	**8**	**9,5**
Duties	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
Type	E	L	E	E	M	E	L	M	M	E
Duration	8,5	8	10	9	8,5	9	10	8	8,2	8

Table 6-12: Numerical Example 4 - Duty's type and duration

Cost per day

A duty on Saturday is multiplied by 1.5 and a duty on Sunday is multiplied by 2

Days	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Cost	1	1	1	1	1	1.5	2

Table 6-13: Numerical Example 4 – Cost per day

First, as an input we choose 43 drivers. Considering our solution algorithm, it is the minimum number of drivers for the construction of a feasible time table. Next we calculate the WHWmin and WHWmax as follows. WHWmin is the minimum feasible working hours per week, it is actually the minimum combination of days and duties, according to labour rules for a single driver. WHWmax on the other hand is the maximum feasible working hours per week for a single driver.

Days	M	T	W	T	F	S	S	Total
WHWmax		7(E)	3(M)	27(L)		8(M)	27(L)	
Max Cost		10	12	15		12*1,5	15*2	85
WHWmin	1(E)	12(M)	2(L)		30(E)	12(M)		
Min Cost	8	9,6	12		8	9,6*1,5		52

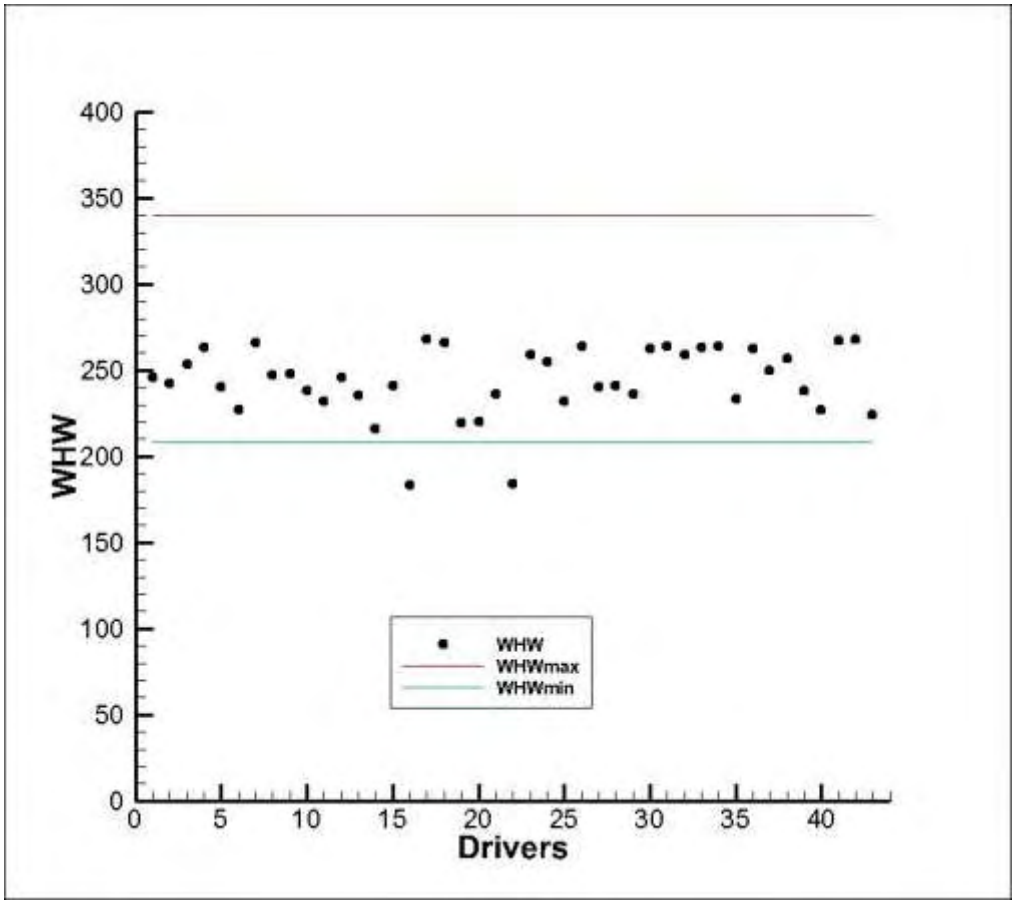
Table 6-14: Numerical Example 4 – WHWmin and WHWmax

$WHW = \frac{WHW_{\min} + WHW_{\max}}{2} = \frac{85+52}{2} = 68.5$ We create a feasible time table WHW=70

Drivers (WHW)					
1	61,6	16	45,75	31	66,1
2	60,5	17	67	32	64,75
3	63,25	18	66,5	33	65,9
4	65,9	19	54,8	34	66,1
5	60	20	55	35	58,44
6	56,84	21	59,2	36	65,5
7	66,68	22	46	37	62,6
8	61,85	23	64,9	38	64,4
9	62,2	24	63,8	39	59,7
10	59,7	25	58,1	40	56,9
11	58,04	26	66,2	41	66,86
12	61,6	27	60,2	42	67
13	58,8	28	60,44	43	56
14	54	29	59,2		
15	60,45	30	65,54		

Table 6-15: Numerical Example 4 – Train Driver's WHW

We managed to create a feasible timetable with 31.7% maximum variance between crew. **CPU time= 17.29 sec. (Intel® Core ™ i5-2430M CPU @ 2.4GHz – RAM 6,00 GB)**



Graph 6-4: Numerical Example 4 – Dispersion of WHW among crew

Chapter 7 CASE STUDY

We focus on crew rostering problem that Greek Railway Administration (Trainose) is facing. Our starting and ending depot is in Athens and we will balance the working hours among the crew. This problem consists of 71 feasible duties that start and end in Athens and Trainose cover those duties with 163 Train drivers. We are going to compare algorithm's results with Trainose's results in August. We are going to simulate this problem by taking into consideration the proportion of days off in August.

At first we created a table of all feasible duties that were generated by trainose in phase one. All duties cover all trips in a specific time period and all duties must be covered by train drivers every day.

ID	Duties	Start	End	Duration	Crew Needed
1	500-501	22.00	07.00	9	5
2	ΕΦΕΔΡΕΙΑ Α/Α	20.00	04.00	8	5
3	4323ΑΙΡ-ΠΕΙΡ.4326/27/30/31/34/4335	17.00	01.00	8	3
4	4530-31-2532-ΕΠ2539	14.30	00.30	10	3
5	1550/57/58/2535	13.30	22.40	9.10	3
6	60-61	17.00	01.00	8	3
7	1552/59/2530/37	14.30	23.40	9.10	2
8	ΕΦΕΔΡΕΙΑ Α/Α	12.00	20.00	8	2
9	ΕΠ.4314ΛΙΟ.+ΕΠ.4415ΑΕΡ.+4018/19/20/21/22/23+ΕΠ.4426 ΛΙΟ&ΕΠ.4327 ΑΙΡ	12.30	20.30	8	1
10	ΕΠ.1538ΣΚΑ+4217/24/25/32/33/40 ΕΠ.1555 ΑΙΡ	8.30	17.00	8.3	2
11	4321ΑΙΡ-ΠΕΙΡ4324/25/28/29/32/33	16.00	24.00	8	1
12	884-55	07.00	16.15	9.15	2
13	1534/41/42/49	05.30	15.00	9.30	2
14	ΕΠ.1532ΟΙΝ+1532ΧΑΛ/1539/40/47	05.30	13.40	8.10	2
15	ΥΠ.4200+4201+4402ΚΙΑΤΟ+ΕΠ.4411 ΛΙΟ.+ΕΦ+ΕΠ1545ΑΙΡ	04.00	12.00	8	2
16	ΕΦΕΔΡΕΙΑ Δ/Η	06.00	14.00	8	2
17	2536-1533	22.10	07.40	9.30	2
18	ΕΦΕΔΡΕΙΑ Δ/Η	22.00	06.00	8	2
19	60-61	17.00	01.00	8	1
20	ΡΕΝ1553ΠΕΙΡΑΙΑΣ/1554/1555/1556/1557ΡΕΝ+1510 ΕΠ.61	15.50	24.00	8	1
21	ΡΕΝ1545Π1546/47/48/49/50/51/52/53ΡΕΝ+ΕΦ.ΠΡΟΑΣΤΙΑΚΟΥ	11.50	20.00	8.10	1
22	ΧΑΛΚ 4421 ΑΕΡ+ΕΦ.-ΕΠ.4428 ΚΙΑΤ 4433	15.30	24.00	8.3	1
23	ΡΕΝ 1557ΠΕΙΡΑΙΑΣ/1558/59/2530/2531Ρ+2534ΟΙΝ2539	16.40	00.40	8	1
24	ΧΑΛΚ+4241/48/49/56/57/64/65	14.30	24.00	9.3	1
25	56-59	13.00	21.15	8.15	1
26	4311ΑΙΡ-ΠΕΙΡ4314/15/18/19/22/23	11.00	19.00	8	1
27	500-501	22.00	07.00	9	1
28	ΡΕΝ2531Π2532/2533/2534/2535/2536/2537Ρ&ΕΦΕΔΡΕΙΑΠΡΟΑΣΤΙΑΚΟΥ	19.50	04.00	8	1
29	ΧΑΛΚ.4245/52/53/60/61	15.00	23.00	8	1
30	58-885	15.00	23.00	8	1
31	58-885	15.00	23.00	8	1
32	1536/43/44/51	06.30	15.40	9.10	1
33	1532ΟΙΝ-1534ΣΠ&ΕΦ&Ρ-Π1541/42/43/44/1545Ρ	04.00	12.30	8.3	2
34	ΧΑΛΚ-ΣΚΑ+4263 ΑΕΡ+ΕΦ	20.30	06.00	9.3	1
35	ΕΛΙΓΜΟΙ ΡΟΥΦ	16.00	24.00	8	1

ID	Duties	Start	End	Duration	Crew Needed
36	ΕΠ.1540ΣΚΑ+4221/28/29/36/37/44+ΕΠ.1557ΑΙΡ	9.30	18.00	8.30	1
37	ΕΛΙΓΜΟΙ ΑΙΡ+ΡΟΥΦ	16.00	24.00	8	1
38	1548/55/56/2533	12.30	21.40	9.10	1
39	ΕΛΙΓΜΟΙ ΑΜΑΞΟΣΤΑΣΙΟΥ	13.00	21.00	8	1
40	50-53	06.00	14.00	8	1
41	ΠΑΡΑΛΑΒΗ 1534/33/36/35ΡΕΝ ΕΦ.	04.00	12.00	8	1
42	1530-4539&1535Ρ-Π1538/39/40/41Ρ	04.00	11.00	7	1
43	ΧΑΛΚ ΣΚΑ+4261 ΑΕΡ+ΕΦ	20.30	06.00	9.3	1
44	50-53	06.00	14.00	8	1
45	ΥΠΗΡ4000/01/02/03/04/05/06/07 ΕΦ.ΑΕΡ+ΕΠ.4220ΛΙΟ+ΕΠ.4311ΑΙΡ	04.00	12.00	8	1
46	ΥΠΗΡ4202/03/08/09/16+ΕΠ.1545ΑΙΡ	04.00	12.00	8	1
47	ΕΦ. Α/Α	04.00	12.00	8	1
48	ΕΦΕΔΡΕΙΑ Δ/Η	14.00	22.00	8	1
49	54-57	11.00	19.00	8	1
50	52 ΑΙΡ. – Θ/Ν	08.30	16.30	8	1
51	51 Θ/Ν - ΑΙΡ	04.00	12.00	8	1
52	1546/53/54/2531	11.30	20.50	9.2	1
53	4309ΑΙΡ-ΠΕΙΡ.4312/13/16/17/20/21	10.30	18.30	8	1
54	52 ΑΙΡ. – Θ/Ν	08.30	16.30	8	1
55	51 Θ/Ν - ΑΙΡ	04.00	12.00	8	1
56	4302/03/06/07/10/11&ΕΦ.Μ.Α.Ι	05.30	13.30	8	1
57	ΕΠ.ΟΙΝ 1530/35/38/45	04.00	13.00	9	1
58	ΥΠΗΡ 4204/05/12/13/20 ΕΠ.1545ΑΙΡ	04.00	12.00	8	1
59	4300/01/04/05/08/09	05.30	13.30	8	1
60	4400ΚΙΑΤΟ+ΕΠΙΒ4405ΑΕΡ.ΕΦ+ΕΠΙΒ4220ΛΙΟ+ΕΠΙΒ1545 ΑΙΡ	04.00	12.00	8	1
61	23500 ΑΙΡ-Θ/Ν	19.00		10	1
62	23503 Θ/Ν-ΑΙΡ	18.00		10	1
63	23500 ΑΙΡ-Θ/Ν	19.00		10	1
64	23503 Θ/Ν-ΑΙΡ	18.00		10	1
65	ΕΛΙΓ.ΜΑΙ Κ ΕΦΕΔΡΕΙΑ	05.00	13.00	8	1
66	ΚΑΤΑΓΡ.ΠΕΤΡ.&ΕΛΙΓ.ΜΑΙ	06.00	14.00	8	1
67	ΕΛ. ΟΙΝΟΗΣ	07.00	17.00	10	1
68	ΕΛ.ΟΙΝΟΗΣ	07.00	17.00	10	1
69	ΕΛ.ΚΕΠ	07.00	15.00	8	1
70	ΘΡΙΑΣΙΟ	03.00	13.00	10	1
71	ΚΑΥΣΙΜΑ	03.00	11.00	8	1

Table 7-1: Case Study – Trainose's feasible Duties

First column represents the train driver's ID. Second column represents the sequence of the trips for every duty, starting from Athens and ending in Athens. Third and fourth columns represent the starting and ending time and at the last two columns we see the duration of every duty and the crew number needed to cover every duty.

Our second step is to categorize our duties into 3 main categories.

4) **Early Duties (E)** start between 06.00 till 13.59. We consider that situation, as normal operation.

5) **Medium Duties (M)** that start from 14.00 till 21.59. The working hours of that category are multiplied by a weight 1.2.

6) **Late Duties (L)** that start from 22.00 till 05.59. The working hours of that category are multiplied by a weight 1.5.

Early Duties							
16	66	67	10	36	49	8	25
40	32	68	50	53	52	9	39
44	12	69	54	26	21	38	5
Medium Duties							
48	24	31	11	23	19	61	2
4	29	22	35	3	62	63	34
7	30	20	37	6	64	28	43
Late Duties							
1	17	15	42	47	57	65	56
18	70	33	45	51	58	13	59
27	71	41	46	55	60	14	

Table 7-2: Case Study – Trainose's duty types

Taking into consideration the duties described in table 7-1 we categorized them depending their starting time and we created a table of Trainose's duty types (Table 7-2)

We assumed that every medium duty's working hours are multiplied by 1.2 and every late duty's working hours are multiplied by 1.5. Also every duty's working hours are multiplied by 1.5 on Saturday and by 2 on Sunday.

Table 7-3 presents Trainose's results

Driver's ID	Working Hours/Month	Driver's ID	Working Hours/Month	Driver's ID	Working Hours/Month	Driver's ID	Working Hours/Month	Driver's ID	Working Hours/Month
1	277,85	34	284,3	67	316,05	100	193,9	133	268,55
2	202,4	35	329	68	218,95	101	139,5	134	283,82
3	286,875	36	284	69	178,4	102	229,5	135	252,895
4	274,075	37	180	70	268	103	103,5	136	299,77
5	290,965	38	192,955	71	263	104	117	137	210,755
6	246,16	39	330,625	72	135,72	105	279	138	295,285
7	350,9	40	221,7	73	312,595	106	272,85	139	178,05
8	118,3	41	158,325	74	148,3	107	238,7	140	220,8
9	180	42	274,55	75	298,3	108	227,475	141	204,68
10	314,64	43	159,55	76	64	109	228,9	142	264,2
11	276,15	44	295,95	77	184,45	110	184,3	143	166,325
12	234	45	236,75	78	309,22	111	212,875	144	255,47
13	240,4	46	222,125	79	234	112	173,6	145	210,29
14	208,64	47	204	80	138,6	113	221,95	146	263,855
15	190,99	48	162,25	81	252,3	114	242,05	147	130,675
16	182,27	49	345,96	82	152	115	261,25	148	345,3
17	288	50	300,02	83	238,9	116	300,15	149	214,07
18	198,465	51	327,95	84	312	117	271,6	150	243,9
19	220,3	52	287,075	85	266	118	276,89	151	203,6
20	207,75	53	178,05	86	280,9	119	133,45	152	219,6
21	251,7	54	182,8	87	131,43	120	286,65	153	180,95
22	256,5	55	100	88	311,16	121	263,395	154	254,7
23	167,04	56	219,1	89	190,4	122	285,75	155	302,26
24	251,65	57	244,85	90	240,875	123	157,65	156	219,1
25	270,7	58	281,325	91	133	124	244,5	157	275,05
26	210,49	59	300	92	236,15	125	281,45	158	267,725
27	292,425	60	199,55	93	95,05	126	239,5	159	275,1
28	179,325	61	268,95	94	302,075	127	303,45	160	148,5
29	322,93	62	160	95	316,55	128	204,825	161	333,99
30	245,17	63	228	96	175,85	129	322,2	162	198
31	207,85	64	182,925	97	263,25	130	221,4	163	108
32	183,275	65	287,645	98	136	131	297,595		
33	124,65	66	182,65	99	336,645	132	282,9		

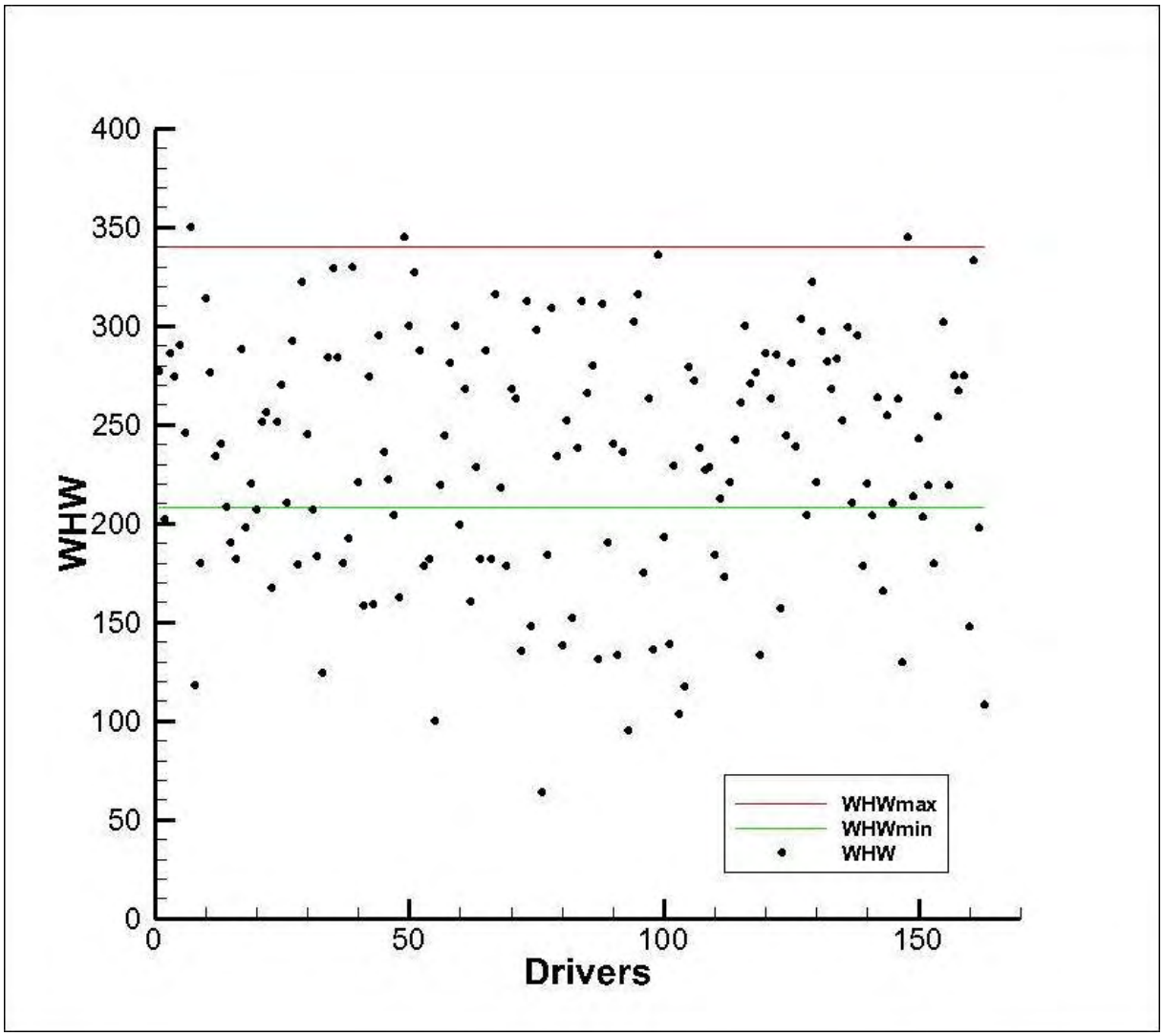
Table 7-3: Case Study – Train Driver's WHW in Trainose's timetable

We calculate the WHWmin and WHWmax as follow. WHWmin is the minimum feasible working hours per week, it is actually the minimum combination of days and duties, according to labour rules for a single driver. WHWmax on the other hand is the maximum feasible working hours per week for a single driver.

Days	M	T	W	T	F	S	S	Total
WHWmax		68(E)	4(M)	65(L)		62(M)	65(L)	
Max Cost		10	12	15		12*1,5	15*2	85
WHWmin	32(E)	43(M)	56(L)		38(E)	19(M)		
Min Cost	8	9,6	12		8	9,6*1,5		52

Table 7-4: Case Study – WHWmin and WHWmax

We create a graph to visualize the dispersion of working hours among drivers



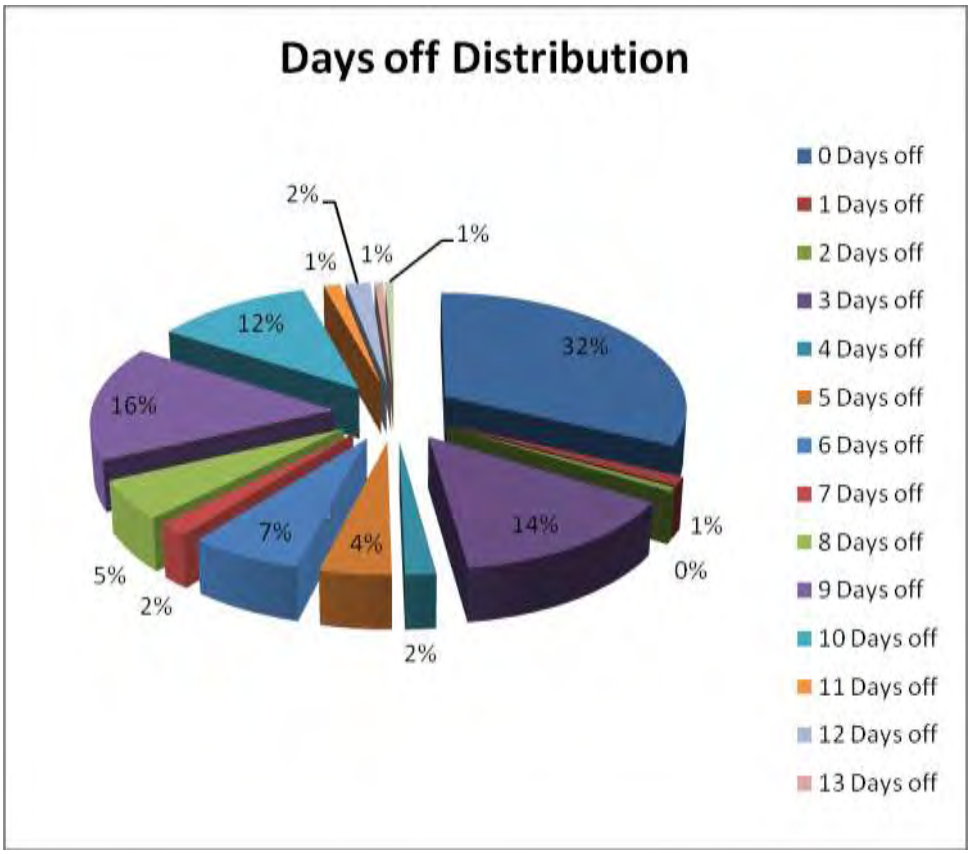
Graph 7-1: Case Study – Dispersion of WHW among crew – Trainose Case

As we can see from the graph, due to the empirical method of rostering taking into consideration days off of train drivers, trainose distributes the working hours unevenly among drivers.

Now we will calculate the number of train drivers by using this formula.

$$No.Drivers \geq No.Shifts * 1.4$$

And then we are going to generate new drivers to cover duties that need more than one driver and days off that were requested from the existing drivers. We created a graph of the proportion of days off vs drivers and we added drivers, to cover the remaining duties, with a simple procedure.



Graph 7-2: Case Study – Proportion of Days off among Drivers – Trainose Case

We assumed that every medium duty's working hours are multiplied by 1.2 and every late duty's working hours are multiplied by 1.5. In addition, every duty's working hours are multiplied by 1.5 on Saturdays and by 2 on Sundays.

Table 7-5 presents Algorithm's Results

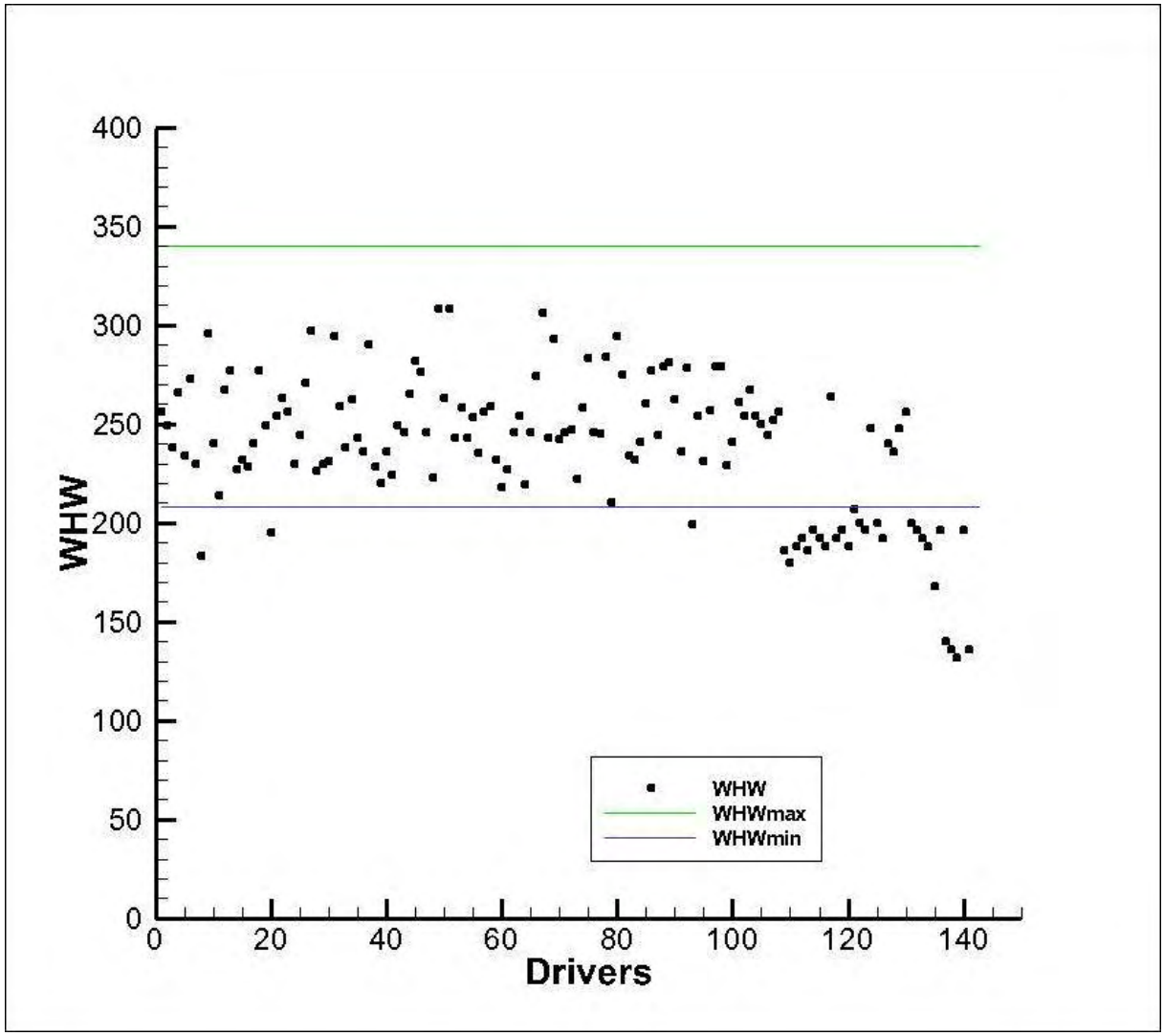
Driver's ID	Working Hours/Month	Driver's ID	Working Hours/Month	Driver's ID	Working Hours/Month	Driver's ID	Working Hours/Month	Driver's ID	Working Hours/Month
1	256,64	34	262,94	67	306,60	100	241,28	133	192,00
2	249,70	35	243,20	68	243,68	101	261,40	134	188,00
3	238,80	36	236,80	69	293,36	102	254,20	135	168,00
4	266,64	37	290,60	70	242,40	103	267,20	136	196,00
5	234,40	38	228,80	71	246,08	104	254,20	137	140,00
6	273,80	39	220,80	72	247,20	105	250,90	138	136,00
7	230,40	40	236,88	73	222,40	106	244,00	139	132,00
8	183,84	41	224,44	74	258,40	107	252,00	140	196,00
9	296,32	42	249,84	75	283,00	108	256,00	141	136,00
10	240,80	43	246,90	76	246,52	109	186,40		
11	214,24	44	265,88	77	245,60	110	180,00		
12	267,20	45	282,16	78	284,20	111	188,00		
13	277,20	46	276,80	79	210,64	112	192,00		
14	227,60	47	246,08	80	294,40	113	186,40		
15	232,30	48	223,36	81	275,48	114	196,00		
16	228,40	49	308,80	82	234,10	115	192,00		
17	240,40	50	263,52	83	232,28	116	188,00		
18	277,52	51	308,00	84	241,20	117	264,68		
19	249,20	52	243,64	85	260,40	118	192,68		
20	195,68	53	258,80	86	277,20	119	196,00		
21	254,40	54	243,60	87	244,64	120	188,00		
22	263,80	55	253,40	88	279,60	121	207,20		
23	256,19	56	235,44	89	281,00	122	200,00		
24	230,24	57	256,60	90	262,40	123	196,00		
25	244,60	58	259,52	91	236,64	124	248,00		
26	271,10	59	232,00	92	278,60	125	200,00		
27	297,80	60	218,10	93	199,60	126	192,00		
28	226,60	61	227,40	94	254,74	127	240,00		
29	230,40	62	246,08	95	231,84	128	236,00		
30	231,40	63	254,60	96	257,44	129	248,00		
31	294,16	64	219,70	97	279,20	130	256,00		
32	259,70	65	246,44	98	279,96	131	200,00		
33	238,44	66	274,60	99	229,60	132	196,00		

Table 7-5: Case Study – Train Driver's WHW in Algorithms's timetable

All in all, as an input, we used imax=100 (The maximum number of train drivers) and we created a feasible time table that covers all duties every day. Then we started generating extra drivers by using a very simple method to cover shifts that need more than one driver,

respectfully to their demands for days off. The extra drivers in Table 7-1 are depicted with green color. At is point we have to say that the generation of extra drivers can be investigated as a separate problem. We can just use the spare capacity of the driver's schedule and generate extra drivers, only in case of infeasibility.

We created a graph to visualize the dispersion of working hours among drivers



Graph 7-6: Case Study – Dispersion of WHW among crew – Algorithm's Results

We can see that our algorithm distributes working hours among crew more evenly while we used for the same problem 12.3% less drivers.

Chapter 8 CONCLUSIONS - FUTURE RESEARCH

In this thesis, we studied an integer programming problem for train driver's rostering and we developed an analytical methodology that balance the working hours among the crew. This methodology is based on the fact that the crew management problem, the generation of feasible duties that cover all trips, was predetermined by trainose.

The proposed algorithm was coded in C++ programming language using libraries from IBM ILOG software and we use CPLEX as a solver. We had a close investigation of some parameters.

We decomposed the duties into three main categories taking into consideration their starting time. So by not allowing drivers to be assigned to duties of the same or lower category for two consecutive days, we secured that every driver has at least 12 hours rest between his duties.

We used weights for Medium and Late duties and by using them as cost parameters in our constraints, we did not only balance the working hours among drivers, but we also created a fair time table for every train driver.

We used Working Hours per Week (WHW) parameter. This parameter is crucial for the construction of a balanced time table of duties and by using empirical values, we managed to reduce computational time.

This algorithm is very flexible because, by determining the appropriate WHW value we can come up with a result that can be implemented in any case.

The algorithm is independent of the number of train drivers and enables us to easily calculate the minimum number of drivers, in any case and build a schedule according to our needs.

Unfortunately that algorithm assumes that all train drivers can be assigned in any duty, which is not applicable to a real world's problem. Furthermore, days off for every driver can only be transferred manually in another week.

Future research should be directed towards the improvement of the existing algorithm. We could possibly use real data as an input and introduce parameter tables that forbid inappropriate drivers to be assigned to inappropriate duties.

Another possible direction is the improvement of that algorithm to create a feasible time table taking into consideration days off (vacation or medical reason) and duties that need to be covered by more than one driver.

Finally we believe that, the creation of another algorithm, that generates feasible duties that cover all trips, is going to minimize the number of duties, hence the global cost. So the closer examination of trainose's crew scheduling problem is considered to be crucial.

At last we reviewed crew scheduling methods in Airline companies. We believe that the Column Generation approach in Rail crew scheduling is very promising. The Train Driver Recovery Problem (TDRP) can be formulated as a *set partitioning problem* and the LP relaxation of the TDRP can be solved with a column generation approach.

Appendix A C++ Implementation of main Algorithm – IBM ILOG (CPLEX)

```
#include <ilcplex/ilocplex.h> // IBM Cplex Library
#include <stdio.h>  // C++ library writes our result in a .txt file
ILOSTLBEGIN

int i,j,d,m;

const int mmax=6; // Artificial Index – Subset of two consecutive days
const int imax=100; // Train Drivers
const int jmax=71; // Feasible Duties
const int dmax=7; // Planning Horizon


int
main (int argc, char **argv)
{

FILE *myfile;

myfile=fopen("teliko_apotelesma.txt","w"); // Results



double T[dmax]; // Cost per day parameter

for (d=0;d<dmax;d++){
        T[d]=1;
        }

T[5]=1.5;
T[6]=2;


double K[jmax]; // Cost per duty parameter

for (j=0;j<jmax;j++){
        K[j]=8;
        }

K[0]=9;
K[3]=10;
K[5]=9.1;
K[7]=9.1;
K[10]=8.3;
```

```
K[12]=9.15;
K[13]=9.3;
K[14]=8.1;
K[17]=9.3;
K[21]=8.1;
K[22]=8.3;
K[24]=9.3;
K[25]=8.15;
K[27]=9;
K[32]=9.1;
K[33]=8.3;
K[34]=9.3;
K[36]=8.3;
K[38]=9.1;
K[43]=9.3;
K[52]=9.2;
K[57]=9;
K[61]=10;
K[62]=10;
K[63]=10;
K[64]=10;
K[67]=10;
K[68]=10;
K[70]=10;


double L[jmax]; // Late duties

for (j=0;j<jmax;j++){
        L[j]=0;
        }

L[0]=1;
L[17]=1;
L[26]=1;
L[16]=1;
L[69]=1;
L[70]=1;
L[14]=1;
L[32]=1;
L[40]=1;
L[41]=1;
L[44]=1;
L[45]=1;
L[46]=1;
L[50]=1;
L[54]=1;
L[56]=1;
```

```
L[57]=1;
L[59]=1;
L[64]=1;
L[12]=1;
L[13]=1;
L[55]=1;
L[58]=1;

double KL[jmax]; // Extra cost for late duties

for (j=0;j<jmax;j++){
        KL[j]=1;
        }

KL[0]=1.5;
KL[17]=1.5;
KL[26]=1.5;
KL[16]=1.5;
KL[69]=1.5;
KL[70]=1.5;
KL[14]=1.5;
KL[32]=1.5;
KL[40]=1.5;
KL[41]=1.5;
KL[44]=1.5;
KL[45]=1.5;
KL[46]=1.5;
KL[50]=1.5;
KL[54]=1.5;
KL[56]=1.5;
KL[57]=1.5;
KL[59]=1.5;
KL[64]=1.5;
KL[12]=1.5;
KL[13]=1.5;
KL[55]=1.5;
KL[58]=1.5;


double E[jmax]; // Early Duties

for (j=0;j<jmax;j++){
        E[j]=0;
        }

E[15]=1;
E[39]=1;
E[43]=1;
```

```
E[65]=1;
E[31]=1;
E[11]=1;
E[66]=1;
E[67]=1;
E[68]=1;
E[49]=1;
E[53]=1;
E[9]=1;
E[35]=1;
E[52]=1;
E[25]=1;
E[48]=1;
E[51]=1;
E[20]=1;
E[7]=1;
E[8]=1;
E[37]=1;
E[24]=1;
E[38]=1;
E[4]=1;

double M[jmax]; //Medium Duties

for (j=0;j<jmax;j++){
        M[j]=0;
        }

M[47]=1;
M[3]=1;
M[6]=1;
M[23]=1;
M[28]=1;
M[29]=1;
M[30]=1;
M[21]=1;
M[19]=1;
M[10]=1;
M[34]=1;
M[36]=1;
M[22]=1;
M[2]=1;
M[5]=1;
M[18]=1;
M[61]=1;
M[63]=1;
M[60]=1;
M[62]=1;
```

```
M[27]=1;
M[1]=1;
M[33]=1;
M[42]=1;


double KM[jmax]; //Extra cost for Medium duties

for (j=0;j<jmax;j++){
        KM[j]=1;
        }


KM[47]=1.2;
KM[3]=1.2;
KM[6]=1.2;
KM[23]=1.2;
KM[28]=1.2;
KM[29]=1.2;
KM[30]=1.2;
KM[21]=1.2;
KM[19]=1.2;
KM[10]=1.2;
KM[34]=1.2;
KM[36]=1.2;
KM[22]=1.2;
KM[2]=1.2;
KM[5]=1.2;
KM[18]=1.2;
KM[61]=1.2;
KM[63]=1.2;
KM[60]=1.2;
KM[62]=1.2;
KM[27]=1.2;
KM[1]=1.2;
KM[33]=1.2;
KM[42]=1.2;
```

```
IloEnv env;

        try {

                IloModel model (env);

typedef IloArray<IloNumArray> IloNumMatrix2x2;
typedef IloArray<IloNumMatrix2x2> IloNumMatrix3x3;

typedef IloArray<IloNumVarArray> IloNumVarMatrix2x2;
typedef IloArray<IloNumVarMatrix2x2> IloNumVarMatrix3x3;
typedef IloArray<IloNumVarMatrix3x3> IloNumVarMatrix4x4;

typedef IloArray<IloRangeArray> IloRangeMatrix2x2;
typedef IloArray<IloRangeMatrix2x2> IloRangeMatrix3x3;



IloCplex cplex(env);



//---------- Decision Variable D for Train Drivers-------------------------------------

        IloNumVarMatrix3x3 Dijd(env,0);
        for (i=0;i<imax;i++){
                IloNumVarMatrix2x2 Djd(env,0);
                for (j=0;j<jmax;j++){
                        IloNumVarArray Dd(env,0);
                        for (d=0;d<dmax;d++){
                                char Ipiresies[70];
                                sprintf(Ipiresies,"Dijd(i%d,j%d,d%d)",i,j,d);
           IloNumVar D(env,0,1,ILOINT,Ipiresies);
           Dd.add(D);

                        }
                        Djd.add(Dd);
                }
                Dijd.add(Djd);
        }
```

```
//------------------------------------------------------------------------
//-------------------------CONTRAINTS-------------------------------------

//------------------------------------------------------------------------
//--------------------(1) Set of constraints: Only one Duty per Driver -----------

IloRangeMatrix2x2 SumDjd(env,0);
        for (j=0;j<jmax;j++){
                IloRangeArray SumDd(env,0);
                        for (d=0;d<dmax;d++){
                                IloExpr expr(env,0);
                                        for (i=0;i<imax;i++){
                                                expr+=Dijd[i][j][d];
                                                }
                                char Ipir_MIX[60];
                                sprintf(Ipir_MIX,"SumDijd(j%d,d%d)",j,d);
                                float LB=1,UB=1;
                                IloRange SumD(env,LB,expr,UB,Ipir_MIX);
                                model.add(SumD);
                                SumDd.add(SumD);
                                expr.end();
                                }
                        SumDjd.add(SumDd);
                        }

//------------------------------------------------------------------------
//------------ (2) Set of constraints: Only one Driver per Duty-------------------------------


        IloRangeMatrix2x2 Sum2Did(env,0);
        for (i=0;i<imax;i++){
                IloRangeArray Sum2Dd(env,0);
                for (d=0;d<dmax;d++){
                        IloExpr expr(env,0);
                        for (j=0;j<jmax;j++){
                                expr+=Dijd[i][j][d];
                                }
                        char Ipir_MIX2[60];
                        sprintf(Ipir_MIX2,"Sum2Dijd(j%d,d%d)",j,d);
                        float LB=-IloInfinity,UB=1;
                        IloRange Sum2D(env,LB,expr,UB,Ipir_MIX2);
                        model.add(Sum2D);
                        Sum2Dd.add(Sum2D);
                        expr.end();
                        }

                        Sum2Did.add(Sum2Dd);
                }
```

```
//-------------------------------------------------------------------------
//-----------(3) Set of constraints: Maximum WHW for every driver-------------


                IloRangeArray Sum3Di(env,0);
                for (i=0;i<imax;i++){
                        IloExpr expr(env,0);
                        for (j=0;j<jmax;j++){
                                for (d=0;d<dmax;d++){
                                expr+=Dijd[i][j][d]*T[d]*KL[j]*K[j]*KM[j];
                                }
                        }
                        char Ipir_MIX3[60];
                        sprintf(Ipir_MIX3,"Sum3Dijd(j%d,d%d)",j,d);
                        float LB=-IloInfinity,UB=77;
                        IloRange Sum3D(env,LB,expr,UB,Ipir_MIX3);
                        model.add(Sum3D);
                        Sum3Di.add(Sum3D);
                        expr.end();
                }

//-----------(4) Set of constraints: No early duty, after early duty-------------

                IloRangeMatrix2x2 Sum4Dmi (env,0);
                for (m=0;m<mmax;m++){
                        IloRangeArray Sum4Di(env,0);
                        for (i=0;i<imax;i++){
                                IloExpr expr(env,0);
                                for (j=0;j<jmax;j++){
                                        for (d=m;d<m+1;d++){
                                expr+=Dijd[i][j][d]*E[j]+Dijd[i][j][d+1]*E[j];
                                }
                        }

                        char Ipir_MIX4[60];
                        sprintf(Ipir_MIX4,"Sum4Dijd(i%d)",i);
                        float LB=-IloInfinity,UB=1;
                        IloRange Sum4D(env,LB,expr,UB,Ipir_MIX4);
                        model.add(Sum4D);
                        Sum4Di.add(Sum4D);
                        expr.end();
                        }
                Sum4Dmi.add(Sum4Di);
                }
```

```
//-------------------------------------------------------------------------
//-----------(5) Set of constraints: No early or medium duty, after medium duty----------

                IloRangeMatrix2x2 Sum5Dmi (env,0);
                for (m=0;m<mmax;m++){
                IloRangeArray Sum5Di(env,0);
                        for (i=0;i<imax;i++){
                        IloExpr expr(env,0);
                                for (j=0;j<jmax;j++){
                                        for (d=m;d<m+1;d++){
        expr+=Dijd[i][j][d]*M[j]+Dijd[i][j][d+1]*M[j]+Dijd[i][j][d+1]*E[j];
                                }
                        }

                        char Ipir_MIX5[60];
                        sprintf(Ipir_MIX5,"Sum5Dijd(i%d)",i);
                        float LB=-IloInfinity,UB=1;
                        IloRange Sum5D(env,LB,expr,UB,Ipir_MIX5);
                        model.add(Sum5D);
                        Sum5Di.add(Sum5D);
                        expr.end();
                        }
                Sum5Dmi.add(Sum5Di);
                }

//-------------------------------------------------------------------------
//-----------(6) Set of constraints: Day off after late duty----------


                IloRangeMatrix2x2 Sum6Dmi (env,0);
                for (m=0;m<mmax;m++){
                IloRangeArray Sum6Di(env,0);
                        for (i=0;i<imax;i++){
                        IloExpr expr(env,0);
                                for (j=0;j<jmax;j++){
                                        for (d=m;d<m+1;d++){

expr+=Dijd[i][j][d]*L[j]+Dijd[i][j][d+1]*L[j]+Dijd[i][j][d+1]*M[j]+Dijd[i][j][d+1]*E[j];
                                }
                        }

                        char Ipir_MIX6[60];
                        sprintf(Ipir_MIX6,"Sum6Dijd(i%d)",i);
                        float LB=-IloInfinity,UB=1;
                        IloRange Sum6D(env,LB,expr,UB,Ipir_MIX6);
                        model.add(Sum6D);
                        Sum6Di.add(Sum6D);
                        expr.end();
```

```
                    }
                Sum6Dmi.add(Sum6Di);
        }

//-------------------------------------------------------------------------
//-----------(7) Set of constraints: No early duty, after early duty for D=7 and D=0------

                IloRangeArray Sum7Di(env,0);
                    for (i=0;i<imax;i++){
                    IloExpr expr(env,0);
                        for (j=0;j<jmax;j++){
                            for (d=dmax-1;d<dmax;d++){
                        expr+=Dijd[i][j][dmax-1]*E[j]+Dijd[i][j][0]*E[j];
                        }
                    }

                    char Ipir_MIX7[60];
                    sprintf(Ipir_MIX7,"Sum7Dijd(i%d)",i);
                    float LB=-IloInfinity,UB=1;
                    IloRange Sum7D(env,LB,expr,UB,Ipir_MIX7);
                    model.add(Sum7D);
                    Sum7Di.add(Sum7D);
                    expr.end();
                    }
//-------------------------------------------------------------------------
//------(8) Set of constraints: No early or medium duty, after medium duty for D=7 and D=0---

                IloRangeArray Sum8Di(env,0);
                    for (i=0;i<imax;i++){
                    IloExpr expr(env,0);
                        for (j=0;j<jmax;j++){
                            for (d=dmax-1;d<dmax;d++){
                        expr+=Dijd[i][j][dmax-
1]*M[j]+Dijd[i][j][0]*M[j]+Dijd[i][j][0]*E[j];
                        }
                    }

                    char Ipir_MIX8[60];
                    sprintf(Ipir_MIX8,"Sum8Dijd(i%d)",i);
                    float LB=-IloInfinity,UB=1;
                    IloRange Sum8D(env,LB,expr,UB,Ipir_MIX8);
                    model.add(Sum8D);
                    Sum8Di.add(Sum8D);
                    expr.end();
                    }
```

```
//-------------------------------------------------------------------------
//------(9) Set of constraints: Day off after Late duty for D=7 and D=0---


                IloRangeArray Sum9Di(env,0);
                        for (i=0;i<imax;i++){
                        IloExpr expr(env,0);
                                for (j=0;j<jmax;j++){
                                        for (d=dmax-1;d<dmax;d++){

expr+=Dijd[i][j][dmax-1]*L[j]+Dijd[i][j][0]*L[j]+Dijd[i][j][0]*M[j]+Dijd[i][j][0]*E[j];
                                        }
                        }

                        char Ipir_MIX9[60];
                        sprintf(Ipir_MIX9,"Sum9Dijd(i%d)",i);
                        float LB=-IloInfinity,UB=1;
                        IloRange Sum9D(env,LB,expr,UB,Ipir_MIX9);
                        model.add(Sum9D);
                        Sum9Di.add(Sum9D);
                        expr.end();
                        }




//-------------------------------------------------------------------------
//-------------------------------------------------------------------------
//--------------------------------Objective Function -------------------------
//-------------------------------------------------------------------------


        IloExpr expr1(env);
for (i=0;i<imax;i++){
        for (j=0;j<jmax;j++){
                for (d=0;d<dmax;d++){
                        expr1+=Dijd[i][j][d]*T[d]*KL[j]*K[j]*KM[j];
                }
        }
}


model.add(IloMinimize(env, expr1));
expr1.end();
```

```
cplex.extract(model);
cplex.exportModel("onoma.lp"); //All Contraint's equation in .lp file



cplex.solve();



                if (!cplex.solve ()){
                        env.error()<<"Faild to optimize LP."<<endl;
                        throw(-1);
                }


                env.out()<<"Solution status = " <<cplex.getStatus()<<endl;
                env.out()<<"Solution value = " <<cplex.getObjValue()<<endl;




for (i=0;i<imax;i++){
                for (j=0;j<jmax;j++){
                        for (d=0;d<dmax;d++){
                         float g = cplex.getValue(Dijd[i][j][d]);
                        if(g!=0) // All none zero values
cout<<"Dijd"<<"("<<i<<","<<j<<","<<d<<")"<<"="<<g<<endl;
                        if(g!=0) fprintf(myfile, "Dijd[%d][%d][%d]=%.0lf\n",i,j,d,g);
//Write the results in .txt file
                        }
                }
        }



        }
        catch ( IloException& e){
                cerr << "concert exception caught:"<<e<<endl;
        }
        catch (...){
                cerr<<"Unknown exception caught" <<endl;
        }
fclose(myfile);
        env.end();
        return 0;
} //End main
```

References

[1] Alberto Caprara, Mateo Fischetti, Paolo Toth, Daniele Vigo, Pier Luigi Guida, "Algorithms for Railway Crew Management" *Mathematical Programming 79 (1997) 125-141*

[2] A.T. ERNST, H. JIANG, M. KRISHNAMOORTHY, H. NOTT and D. SIER "An Integrated Optimization Model for Train Crew Management", *Annals of Operations Research 108, 211–224, 2001.*

[3] Raymond S K Kwan. "Case studies of successful train crew scheduling optimization" *Theory and Applications (MISTA 2009) 10-12 August 2009, Dublin, Ireland.*

[4] Alberto Caprara, Mateo Fischetti, Paolo Toth, Daniele Vigo, Pier Luigi Guida , "Solution of Large – Scale Railway Crew Planning Problems: The Italian Experience".

[5] M Lezaun, G P´erez and E S´ainz de la Maza, "Staff rostering for the station personnel of a railway company" *Journal of the Operational Research Society (2010) 61, 1104 – 1111*

[6] "Personalized Crew Rostering at Netherlands Railways" *Geert de Pont, University of Tilburg*

[7] Ann Sau King Kwan "Train Driver Scheduling", *for the degree of Doctor of Philosophy, The University of Leeds, August 1999.*

[8] Anneke Hartog1, Dennis Huisman2,3, Erwin J.W. Abbink2, and Leo G, Kroon, "Decision Support for Crew Rostering at NS" *Econometric Institute Report EI2006-04*

[9] Chi-Kang LEE Chao-Hui CHEN, “SCHEDULING OF TRAIN DRIVER FOR TAIWAN RAILWAY ADMINISTRATION”, *National Cheng Kung University*

[10] Natalia J. Rezanova, David M. Ryan, “Solving the Train Driver Recovery Problem”, *Extended Abstract*

[11] RAMON M. LENTINK, MICHIEL A. ODIJK & ERWIN VAN RIJN ERIM “CREW ROSTERING FOR THE HIGH SPEED TRAIN” *ERS-2002-07-LIS, February 2002*

[12] Ignacio Eduardo Laplagne, “Train Driver Scheduling with Windows of Relief Opportunities”, ”, *for the degree of Doctor of Philosophy, The University of Leeds, January 2008.*

[13] Leo Kroon, Matteo Fischetti “Scheduling Train Drivers and Guards: the Dutch “Noord-Oost” Case”, *Proceedings of the 33rd Hawaii International Conference on System Sciences – 2000*

[14] RALF BORNDORFER, UWE SCHELTEN, THOMAS SCHLECHTE, STEFFEN WEIDER, “A Column Generation Approach to Airline Crew Scheduling” *ZIB-Report 05-37 (August 2005)*