



UNIVERSITAT AUTONOMA DE BARCELONA
MASTER THESIS

Staff Scheduling in Ground Handling Company



Student:
Elena Popova

Supervisors:
Dr. Miquel Àngel Piera
Dr. Liana Napalkova

September 2015

Abstract

This master thesis aims to analyze ground handling scheduling task in large scale airline operations in order to collect results which might be used in contribution to the operational efficiency of ground handling companies.

This work is focused on designing and developing of evolutionary computation algorithm in Matlab for solving ground handling staff scheduling problem, which consists in finding the schedule for all ground handling agents, such that the objective function is minimized and all the constraints are satisfied. The objective function is built in a way that the number of unserved by some reason aircrafts is as small as possible, the workload is distributed equally and situations when the agent is sent successively to gates located far from each other are mostly avoided. The constraints that were taken into account are time windows of the tasks, working hours of the staff and transportation time between the gates.

Since many factors could affect the processes which are performed during the turnaround, it is not an easy task to make a full simulation, but in this paper was made an attempt to take at least the main variables into consideration.

The analysis of problem solving methods was done in the theoretical part of the work in order to choose the one, which is the most appropriate to the specifics of formulated problem.

Finally, two cases corresponding to two different implementations were considered in this thesis and their comparative analysis was conducted in order to allow more exhaustive exploration and validation of the optimization method.

Table of contents

1	Introduction	4
1.1	State of the Art	4
1.1.1	Economic Aspects of Ground Handling	4
1.1.2	Impact of Turnaround Performance on Flight Delays	5
1.1.3	Manual vs. Automated Scheduling	6
1.1.4	Commercial Solution (INFORM)	7
1.2	Objectives	8
1.3	Methodology	8
1.4	Structure of the Thesis	8
2	Analysis of the Ground Handling Scheduling Problem	9
2.1	Aircraft Ground Handling Activities	9
2.1.1	Deplaning and Boarding	10
2.1.2	Baggage Handling	10
2.1.3	Cargo Handling	10
2.1.4	Aircraft Refueling	11
2.1.5	Cabin Service	11
2.1.6	Catering	11
2.1.7	Passenger Services	11
2.1.8	Water Supply and Lavatory Drainage	12
2.1.9	Supplies of Power, Air-Conditioning and Compressed Air	12
2.1.10	Aircraft De-icing	12
2.1.11	Pushback Operations	13
2.2	The Impact of Uncertainty on Ground Handling Operations	14
2.3	The Planning Horizon	15
2.4	Airport Collaborative Decision Making (A-CDM)	16
2.4.1	A-CDM Milestones Approach	17
2.5	System Wide Information Management (SWIM)	17
3	Mathematical Formulation of the Problem	20
3.1	Classifying Routing and Scheduling Problems	20
3.2	Notation	21
3.3	Input Parameters	21
3.4	Objective Function	22
3.5	Decision Variables	22
3.6	Constraints	23
4	Overview of Problem Solving Methods	24
4.1	Exact methods	24
4.2	Heuristics	25
4.3	Metaheuristics	25

4.3.1	Simulated Annealing	26
4.3.2	Tabu Search	27
4.3.3	Particle Swarm Optimization	28
4.3.4	Ant Colony Optimization	29
4.3.5	Evolutionary Algorithms	31
4.3.5.1	Genetic Algorithm	31
4.3.5.2	Evolutionary Programming	32
4.3.5.3	Evolution Strategies	33
5	Decision Support Tool	35
5.1	Algorithm Outline	35
5.2	Initial Population	36
5.2.1	Case 1	36
5.2.2	Case 2	36
5.3	Termination Criterion	36
5.4	Tournament Selection	36
5.5	Genetic Operations	36
5.5.1	Case 1	36
5.5.2	Case 2	37
5.6	Fitness Function	37
5.6.1	Case 1	37
5.6.2	Case 2	37
6	Case Studies	38
6.1	Input Data	38
6.2	Case Study 1	41
6.2.1	Presentation of Results	41
6.2.2	Computational Time Analysis	44
6.3	Case Study 2	46
6.3.1	Presentation of Model Output	46
6.3.2	Computational Time Analysis	48
7	Analysis of Results	50
8	Conclusions and Future work	52
8.1	Conclusions	52
8.2	Future Work	53
	Annex A	56

Chapter 1

Introduction

1.1 State of the Art

During the ground time at the airport, an aircraft requires different handling services, e.g. the unloading and loading of baggage, the cleaning of the aircraft cabin, fuel and water supply, etc. Previously those operations were performed by a division of the airport.

The Directive 96/67/EC dating from October 1996 [1] has opened up the ground handling market to competitors to prevent the monopoly of EU airports. This resulted not only in reducing the costs of ground handling for the airlines but also in improving of the quality of the services. According to the Directive, in major EU airports, which have more than 2 million passengers or 50000 tonnes of freight per annum, at least two ground handling suppliers must be available for certain service categories. At least one of the suppliers must be independent from the airport and the carrier.

The aircraft turnaround is crucial for airline schedule adherence, for high customer satisfaction, and economic productivity. Ground staff scheduling is an important area of research, since even small improvements in staff scheduling could translate into large savings for airline companies.

This work presents model and algorithms for general optimization and decision problems arising within ground handling staff scheduling.

1.1.1 Economic Aspects of Ground Handling

Recently, many airports and airlines made the transition in ownership towards privatization, which caused the airport business environment become more competitive. Airports are in competition to attract airline routes, while airlines are trying to cut their costs. "We only make money off our planes when they are in the air" says Chris Wahlenmeier, vice president of ground operations in Southwest Airlines [2].

The airlines are eager to turn their planes round fast and get them back in the air as soon as possible. An example could be Ryanairs' attitude towards hold luggage. Hold luggage takes time to load and unload; it also adds weight to the plane which results in higher price for fuel per trip. Less hold luggage means less time for loading/unloading and refueling, therefore less time to be spend in the airport (not to mention the price of the fuel). In this case Ryanair charges for hold luggage rather to reduce the turnaround time and therefore ground handling costs than to get some "extra" money from customers.

Shorter turnaround time results in more trips per plane with the added benefit of getting more trips out of airline staff. Therefore ground handling companies are urged to increase cost effectiveness and to deliver faster and more reliable service. The monetary value of ground handling services accounts for about 5 to 8 percent of the airline ticket, depending on the type of airline being used. The global market for ground handling is now estimated to be worth over

USD80 billion per annum according to its trade association. By comparison, the airline industry turned over around USD789 billion in 2014.

Since ground handling crews are located at the airport all the time, it is possible to call in extra hands if needed. This has made the need for a good planning solution less pressing for ground handling companies than airlines in the past. However, even small improvements in staff planning of ground handling company could translate into large savings.

1.1.2 Impact of Turnaround Performance on Flight Delays

An airline schedule is rarely implemented as planned. It is often disrupted due to bad weather conditions, aircraft breakdowns, crew delays, insufficient ground operation performance, etc.

Figure 1.1 shows an average number of departures a day within a month from January 2011 to April 2015 in Europe.

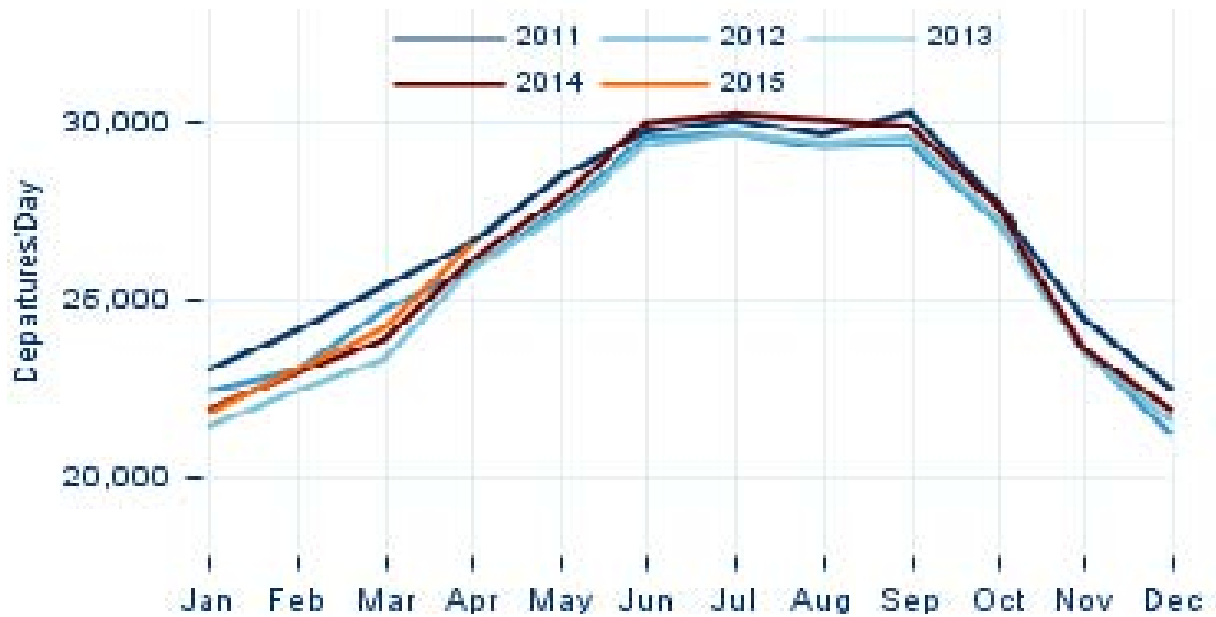


Figure 1.1: Traffic 2011-2015 [3]

Statistics shows that an average delay per delayed flight in April 2015 appeared to be 27 minutes. 40% of delayed flights were delayed on departure and 39% of flights were delayed on arrival (with a delay time > 5 minutes) [3]. More statistics concerning worldwide flight delays could be found on www.flightstats.com/go/Media/stats.do/

Reasons for flight delays can be assigned to five main categories, which cover up to 90% of potential flight delays: Late-arriving Aircraft, Air Carrier, National Aviation System, Extreme Weather and Security [4].

- Late-arriving Aircraft: the previous flight using the same aircraft arrived late, causing the present flight to depart late.
- Air Carrier: the cause of the cancellation or delay was due to circumstances within the airline's control (e.g. maintenance or crew problems, aircraft cleaning, baggage loading, fueling, etc.).
- National Aviation System: delays and cancellations attributed to the national aviation system, such as heavy traffic volume or air traffic control.

- Extreme Weather: significant meteorological conditions (actual or forecasted), such as tornado, blizzard or hurricane.
- Security: delays or cancellations caused by evacuation of a terminal, re-boarding of aircraft because of security breach, inoperative screening equipment, etc.

On Figure 1.2 is given the distribution of delay causes, according to data of 14 major airlines in USA.

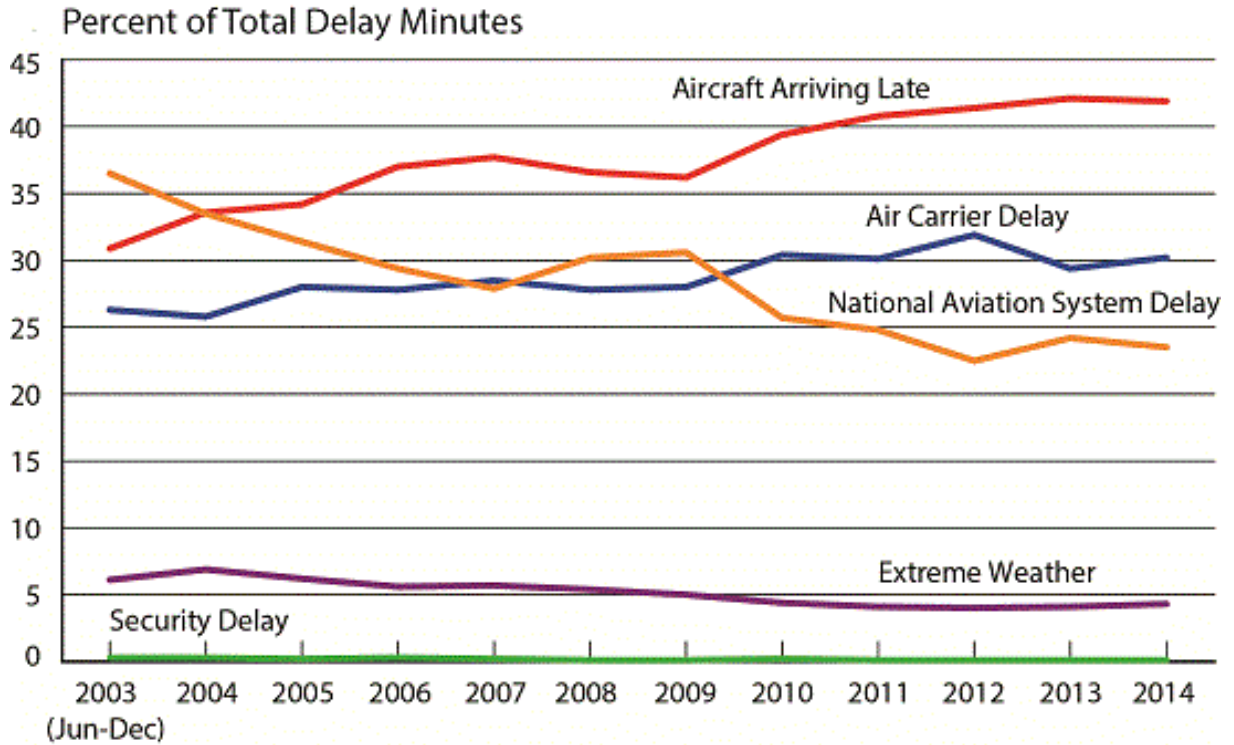


Figure 1.2: Delay cause by year [4]

As it could be seen from Figure 1.2, on one hand, the ground handling activities (which are included in category "Air Carrier Delay") have a significant influence onto the air transport performance, i.e. the delay in providing ground handling operations may result in delay of the flight.

On another hand, the scheduled turnaround process is always disturbed if the airplane does not arrive at the allocated gate on time. Ground handling agents have to consider the possible delays to ensure reliable ground processes aiming to not let this delay propagate into the following flight schedule.

This means that ground handling performance and aircraft performance in terms of delays are strongly interrelated. That is another reason why ground staff scheduling plays an important role.

1.1.3 Manual vs. Automated Scheduling

Ground staff scheduling is a very complex task. Planning often involves hundreds of employees, thousands of work tasks per week and a large number of constraints. Manual planning is tedious and error-prone task, and it often takes planners several hours or even days to prepare staff plans by hand. Therefore the need for automated scheduling systems is becoming significant.

Difficulty which appears while designing the automated system is that not all the constraints can be modeled and not all the information can be put into the optimization model. For exam-

GroundStar is currently used by over 50 organizations in more than 150 airports worldwide.

Scheduling software is nowadays required to flexibly adapt to changes in work rules, problem characteristics and sizes of operations. Clearly, the development of advanced planning systems is a costly investment. Therefore many airlines, airports and ground handling companies resort to generic software packages, such as the one presented by INFORM. However, the use of generic software has a drawback: the solution is often not fully adapted to the realities of that particular ground handling company using it. On one side, scheduling system could include "extra" features; on another side, it could lack some. Therefore ground staff scheduling problem remains actual up to this day.

1.2 Objectives

The objective of this master thesis is to analyze and solve ground handling scheduling problem in large scale airline operations in order to collect results which might be used in contribution to the operational efficiency of ground handling companies.

In order to achieve this objective, the following tasks have been formulated:

1. Analyze different aspects of the ground handling staff scheduling problem.
2. Formulate the mathematical model.
3. Investigate problem solving methods.
4. Develop programming code for ground handling staff scheduling problem with the use of evolutionary algorithms.
5. Perform comparative analysis of results using Design of Experiments (DOE).

1.3 Methodology

In this master thesis ground handling staff scheduling problem was solved with the use of evolutionary strategy algorithm with local heuristics. Programming code was developed in Matlab, Release R2012a.

1.4 Structure of the Thesis

The thesis is organized as follows. Chapter 1 is an introductory part which highlights the importance of ground handling staff scheduling problem and reviews its existing solutions. Chapter 2 provides an overview of the ground handling activities and modern technologies used for exchanging information between aviation partners. Chapter 3 is devoted to the mathematical model. Here the optimization problem is presented by an objective function, which should be minimized given the set of constraints. Chapter 4 reviews the problem solving methods and explains why in this work evolutionary algorithms were selected. Chapter 5 provides information about the chosen features of the algorithm. Chapter 6 is dedicated to description of input data and case studies. In the last chapters 7 and 8 analysis of the results and conclusions are given.

Chapter 2

Analysis of the Ground Handling Scheduling Problem

2.1 Aircraft Ground Handling Activities

Ground handling is a common term to name tasks that should be performed at the aircraft during its turnaround – the time between its arrival to the gate and its departure to the next flight. These tasks can be distinguished by ramp operations and passenger services. Tasks on the ramp include refueling, aircraft cleaning, water supplying, lavatory drainage, baggage and cargo handling, catering, towing with pushback tractors and deicing. Passenger services mainly refer to providing of check-in and boarding services, staffing the transfer counters, customer service counters and airline lounges.

Figure 2.1 shows ground support systems and mobile equipment of Boeing 777 for a typical turnaround.

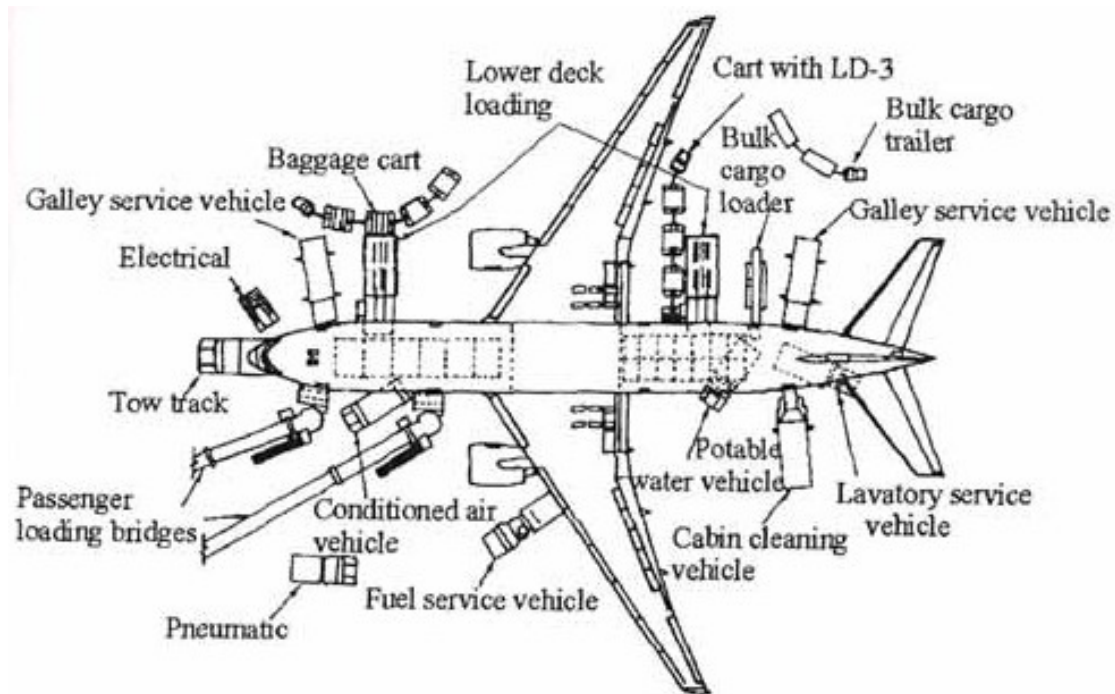


Figure 2.1: Boeing 777 being serviced during a turnaround with the help of ground support systems and mobile equipment [5]

In the next subsections each of GH operations will be reviewed in details in order to give more thoroughful view on particular tasks that ground handlers are responsible for.

2.1.1 Deplaning and Boarding

Disembarking and boarding can be provided by means of:

- stairs carried by the aircraft
- mobile stairs
- passenger bridges.

A combination is also possible, e.g. a bridge attached to the aircraft door behind the crew cabin and mobile stairs for the rear exit.

There are many advantages of using passenger bridges over the stairs. Firstly, passengers can leave/board the aircraft more quickly. Secondly, passengers reach their aircraft safely without getting lost, since their flow path is straight-forward. Thirdly, passengers are protected against bad weather. At last, while the aircraft is being boarded or deplaned, other servicing activities, including aircraft refueling and water supply, can be carried out simultaneously. In this case the movement of the servicing vehicles across the apron is not obstructed. The safety of passengers is also ensured as contact between them and servicing vehicles is avoided.

However, passenger bridges are more costly than mobile stairs and airline companies are charged for using them. That is the reason why low cost carriers usually prefer mobile stairs or even the aircraft's own stairs.

2.1.2 Baggage Handling

Baggage handling is becoming a critical activity in ground handling. One of the criteria for quality of service of the airport is the time passengers have to wait for baggage after disembarking the aircraft.

At small airports most of the activities connected with baggage handling can be carried out manually. The baggage is loaded on the baggage cart either directly or from the belt conveyor and dispatched to the aircraft, where it is loaded in the aircraft cargo holds. At large airports with the departure of several flights at the same time, sorting of baggage must be provided. Sorting can be done manually, when the baggage is picked up from the carousel by the handlers and loaded on the carts, or automatically. Automated sorting uses baggage tags with bar codes, which are automatically printed when passengers check-in.

The baggage handling process still has a considerable human content, leading to many health and safety concerns. Also because of the human factor mishandling can occur, leading to a certain number of lost or mislaid bags, which damages the reputation of the airport.

2.1.3 Cargo Handling

Most of the freight is transported through only few large hubs: Memphis, Hong Kong, Tokyo, Frankfurt, etc. Even though there is an impressive range of available freighters, more than 60 per cent of world freight is taken on passenger aircrafts.

Cargo handling includes movement of goods from landside to airside and vice-versa, or from aircraft to aircraft. There are 3 levels of mechanization to be provided for cargo handling:

- manual: manpower plus fork lift trucks
- semi-mechanized: roller beds or conveyors
- fully mechanized: Elevating Transfer Vehicles (ETV), Automatic Storage and Retrieval Systems, Transfer Vehicles.

The fully mechanized approach only really works with high volumes of containerized freight and in a setting that can guarantee good maintenance skills. Even so, the whole terminal can come to a halt if an ETV breaks down. Transfer Vehicles are powered roller conveyors mounted on a carriage with bogies which are electrically driven along rails in response to a command system. Mechanization is much more expensive in terms of first cost, but reduces the labor requirement, causes less handling damage and less risk of mishandling.

2.1.4 Aircraft Refueling

Distribution of fuel from the fuel storage at the airports into the airplanes may be executed in two ways: by tankers or by hydrants. Both of the systems have their advantages and disadvantages.

The principal advantage of the fixed fuel delivering system is a reduction of operating costs and an increase of safety during aircraft refueling. There are no tankers on the apron with a large quantity of highly inflammable fuel, but only a dispenser with fuel in hoses from the hydrant to the airplane and in the filters. The fuel supply to a single Boeing 747 usually amounts to 100 m^3 or 80 tons of fuel. With a fixed fuel delivering system, in the majority of cases, only one worker is needed to perform refueling. In comparison, for the same operation, two or three tankers with operators are required.

The investment costs of constructing the hydrant system are several times higher than those of a purchase of several tankers. Nevertheless, its operating costs are lower and its service life is longer. Even though a hydrant system of fuel distribution has been constructed in the airport, it is necessary to have available several tankers in case of failure or maintenance of a part of the hydrant system, so that fuel can still be supplied.

2.1.5 Cabin Service

Cabin services ensure passenger comfort. There are basically three levels of cleaning services of the aircraft: quick transit cleaning, overnight cleaning and deep cleaning.

The types of services depend on the duration of turnaround. The tasks in a typical quick transit cleaning involve: (1) seat cleaning, (2) seat pocket cleaning, (3) galley cleaning, (4) toilet cleaning and replenishment, (5) floor cleaning and (6) blanket management. Overnight cleaning is more thorough and in addition to the above processes includes: (1) floor vacuuming, (2) window cleaning, (3) stowage cleaning and (4) cleaning of the cabin crew resting area. Deep cleaning includes all the above tasks but in a more extensive manner [6].

2.1.6 Catering

Catering includes unloading of unused food and drinks from an aircraft, and loading of fresh food and drinks for passengers and aircraft crew. Airlines meals are typically delivered in trolleys. Empty or trash-filled trolleys from the previous flight are replaced by fresh ones. Meals are usually prepared on the ground in order to minimize the amount of preparation (apart from chilling or reheating) required in the air.

While some airlines provide their own catering, others have outsourced their catering to third-party companies.

2.1.7 Passenger Services

Passenger handling includes following services:

- Providing check-in counter services for the departing passengers, i.e. accepting baggage that has to go in the aircraft's cargo hold and issuing boarding passes. Some low cost airlines tend to use online check-in, which is the process when the passengers confirm

their presence on a flight via Internet and typically print their own boarding passes. By Ryanairs' policy, passengers who have lost/forgotten their printed boarding passes are required to pay a Boarding Card Reissue Fee in order to receive a new one at the airport.

- Providing gate arrival and departure services. The handling agents are required to meet a flight on arrival and guide passengers to baggage claim areas and terminal exit, as well as provide departure services including boarding passengers, closing the flight, etc.
- Assistance of arriving and departing VIP, unaccompanied children, disabled and elderly passengers from the point of arrival at the terminal through the whole process to the aircraft boarding
- Staffing the Transfer Counters, Customer Service Counters, Airline Lounges, Lost and Found, etc.

2.1.8 Water Supply and Lavatory Drainage

Aircraft are supplied with potable and non-potable water for lavatory sink use through temporary connections at the airports. Ground handlers typically board aircraft water using designated watering points (facilities where water is transferred from a water supply to an aircraft, including water cabinets, carts, trucks and hoses).

Each aircraft equipped with a bathroom or lavatory needs to expend its waste somehow. After the aircraft arrives it is the lavatory agent's job to flush the lavatory system.

2.1.9 Supplies of Power, Air-Conditioning and Compressed Air

Most aircrafts can meet their energy requirements on a ramp with a use of an Auxiliary Power Unit (APU). Advantages of such a solution are the independence of the ground source and savings on time required to connect/disconnect the ground source. However, the total costs of APU operations, including maintenance costs, are often higher than the charges for using external ground sources. Other disadvantages are low efficiency of the APU (only about 30%), and high noise and environmental pollution due to exhaust gases. At Copenhagen airport, APUs have to be turned off within 5 minutes after the plane docks and at Zürich airport this time period is reduced to 30 seconds. After that, the aircraft has to be supplied with an external electrical power.

If there is no ground support system supplying air-conditioning, the pilot cannot avoid using APU in order to provide ventilating, heating or cooling in the cabin. Air-conditioning has to be provided even in mild latitudes. Due to the high concentration of people aboard, cooling of the cabin has to be assured when the temperature rises above 10 °C. When the temperature is between 4.4–10 °C air has to be circulated, and when it drops below 4.4 °C the cabin has to be heated. Central distribution systems can supply air of -6.5 °C in summer and +66 °C in winter. In order to start the engines compressed air of 230 C° is supplied [7].

2.1.10 Aircraft De-icing

It is dangerous for an aircraft to take off while it is covered by snow or ice. Creating sufficient lift requires high values of the lift coefficient. Research has shown that even a 0.5 mm layer of ice covering the whole upper area of the wing can decrease the maximum lift coefficient by up to 33%. Because of this in the majority of countries it is strictly required that de-icing is carried out prior to take-off.

According to the Association of European Airlines recommendations [8] there are two different procedures for the aircraft surfaces treatment:

- De-icing is a procedure by which frost, ice, slush or snow is removed from an aircraft in order to provide clean surfaces
- Anti-icing is a precautionary procedure which provides protection against the formation of frost or ice and accumulation of snow or slush on treated surfaces of the aircraft for a limited period of time.

For the de-icing procedures, fluids based on glycol are mostly used. However, if an aircraft is covered with thick layers of snow, slush, etc. some airline companies are using an effective and cheap "mechanical" method – snow is simply brushed away.

2.1.11 Pushback Operations

Pushback is a procedure during which an aircraft is pushed backwards away from an airport gate by external power. Pushbacks are carried out by special, low-profile vehicles called pushback tractors or tugs.

Although many aircraft are capable of moving themselves backwards on the ground using reverse thrust, the resulting jet blast may cause damage to the terminal buildings or equipment. Engines close to the ground may also blow sand and then suck it into the engine, causing damage to it. A pushback is therefore the preferred method to move the aircraft away from the gate.

Figure 2.2 shows a typical turnaround of a commercial aircraft at the airport. As we can see, many tasks are performed simultaneously. The considered turnaround time is 60 minutes, but this time can be much shorter in the case of regional jets. Usually, ground handling operations are supposed to be finished within 20 minutes which leaves no room for errors.

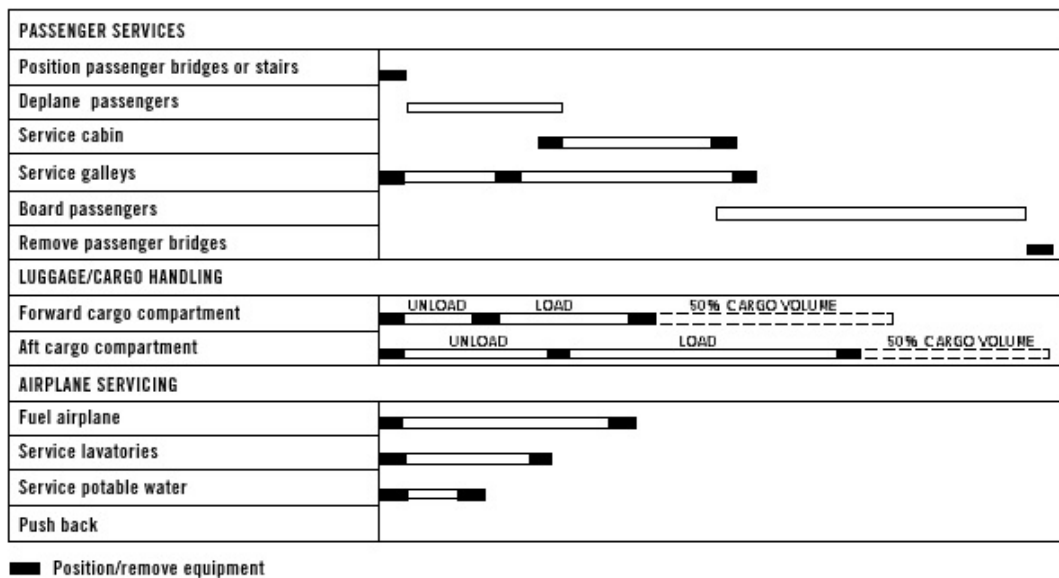


Figure 2.2: Example of ground activities for a commercial aircraft [9]

2.2 The Impact of Uncertainty on Ground Handling Operations

The tighter the operational schedule is, the more important is the operational management. Figure 2.3 shows that sometimes ground handlers have to deal with uncertainty. If a breakdown occurs because of an equipment failure, the operation manager has to make a choice depending on the importance of the interrupted operation. If the flight does not rely on this operation, the manager may decide to cancel or to shorten it. For example in case of freight/mail loading, delays of the aircraft may cost more money than loading can make. The problem considered on Figure 2.3 is the breakdown of the cargo loader.

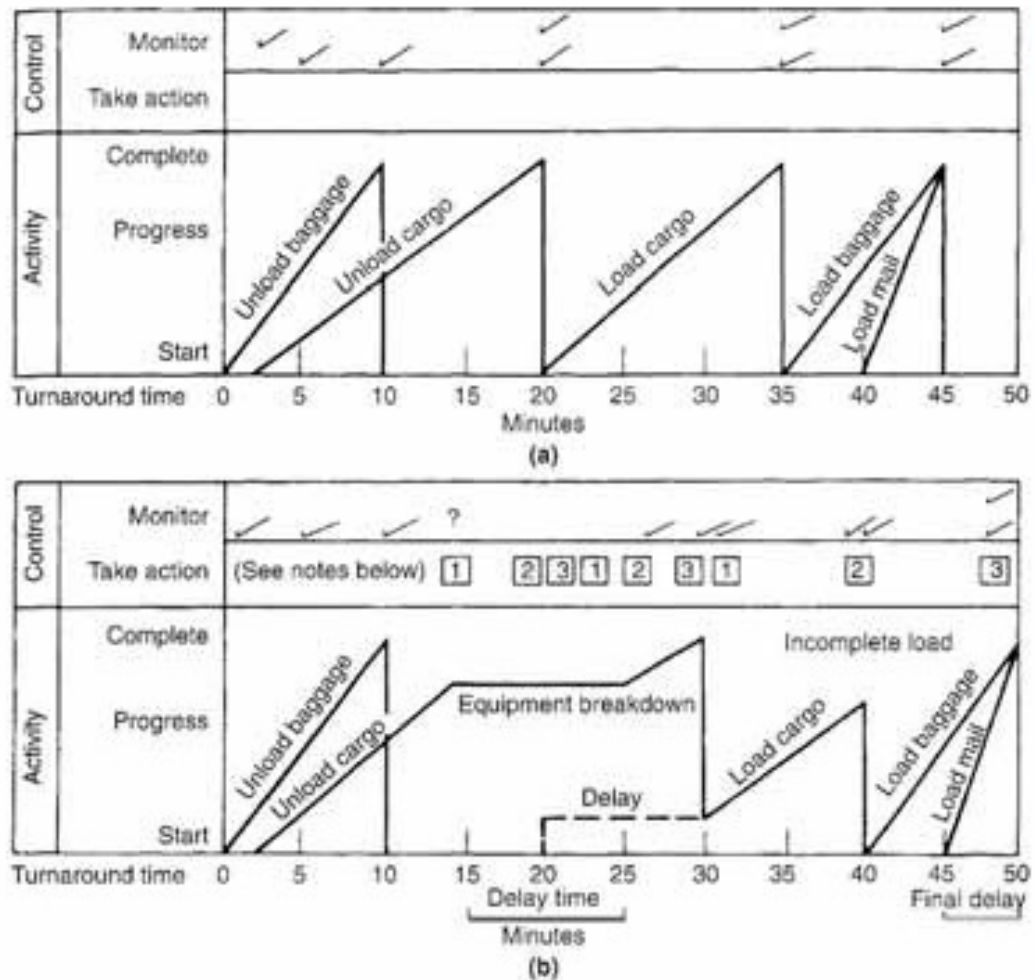


Figure 2.3: Effect of breakdown and delay on apron dispatch [10]

Figure 2.3 (a) represents normal activity, when no control action is required. Figure 2.3 (b) represents a delay appeared through a breakdown. In this case control is required and is performed through the following actions:

- Assess the nature of the problem and realize how much time the problem will take to be sorted out.
- Take corrective action: call equipment base and ask engineer to come to the aircraft or call up a replacement loader.
- Advise all other activities which will be affected by the breakdown and give them instructions as necessary (e.g., notify movement control of a delay, tell passenger service to delay boarding, etc.).

2.3 The Planning Horizon

Ground handling staff scheduling gives rise to a number of challenging optimization problems. The workload is naturally represented by a list of work tasks, which should be performed by employees of appropriate qualification. It is typical to classify staff scheduling problem according to temporal dependencies into strategic, tactical, operational and real-time planning (Figure 2.4). These stages are strongly interrelated and different optimization problems have to be solved at each of them.

Strategic planning, also known as demand planning, is the most distant from the day of operation and is mostly associated with long-term decision making. It could significantly change the size and composition of the workforce by hiring or firing personnel, equipment acquisitions and scenario analysis for bidding on new contracts.

Tactical planning is concerned with mid-term problems that determine availability on the day of operation from a pool of resources that has been predetermined during the strategic planning step. During tactical planning shift duties for future time period, such as next month or next season, are generated. Therefore this phase is sometimes referred to shift planning.

Operational planning deal with an actual day of operation and it mainly covers daily rostering problems in task scheduling. These could be calculated at the beginning of the day or few days before the day of operation. At this phase, the demand and the resource availability are fixed. The problem is then to assign the work to available personnel as efficiently as possible under the set of constraints specified by the previous planning steps. Some of staff management decisions, such as shift swaps, sickness or overtime handling, may be viewed as operational.

Real-time planning, also known as dynamic planning, is the final stage of staff scheduling. It is associated with adapting an existing plan for the day of operation to handle failures that may occur during the day, such as flight delays or equipment breakdowns.

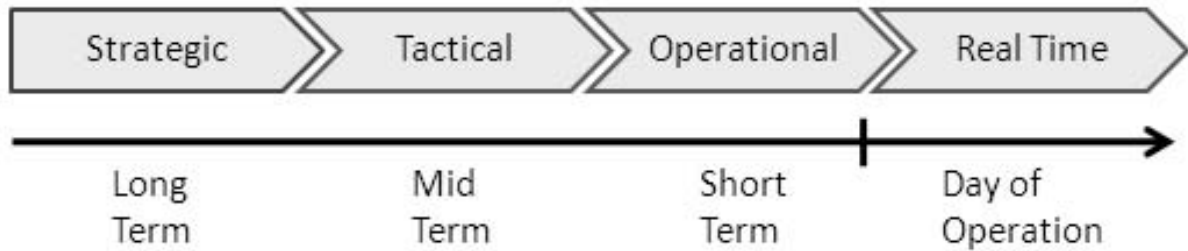


Figure 2.4: Planning timeline

In this thesis operational ground staff scheduling will be considered, the estimation of the workload and the generation of staff timetables will be done at the beginning of the day or several days before the day of operation. This means that the workload for the planning period is calculated from a fixed flight schedule and known Service Level Agreements (SLAs) between airlines and ground handling companies. SLA set the level of quality of the handling services, including strictly definition of their order and time windows. In the event of failure to meet specified standards, ground handling companies are charged compensation.

As for example, the performance standards for delivery times of incoming luggage at Manchester Airport, agreed between airlines and ground handling agents are the following:

Aircraft seat numbers	Maximum delivery times (in minutes)	
	First bag	Last bag
180 or more	25	40
85 to 179	20	30
40 to 84	15	25
Up to 40	12	15

where the time refers to minutes after ATA (Actual Time of Arrival).

Source: www.caa.co.uk/docs/5/ergdocs/ccreportma/chapter5.pdf.

Real-time aspects like short-term schedule changes, flight delays or personnel sickness could also be taken into account by rerunning the algorithm with new input data or by introducing manual changes in the schedule.

2.4 Airport Collaborative Decision Making (A-CDM)

The activities of airlines and airports are complementary in nature but the industry is in need of better coordination between all the aviation partners. Airport Collaborative Decision Making (A-CDM) is a concept aiming to improve the operational efficiency of all airport operators by reducing delays, increasing the predictability of events during the process of a flight and optimizing the utilization of resources. This aim is to be achieved via improved real time information sharing between airport operators, ground handlers, airlines and air traffic control and via adopting coordinated operational procedures, automatic processes and user friendly tools.

Figure 2.5 illustrates the key objectives of each of the partners involved in the Collaborative Decision Making.



Figure 2.5: Partners involved in A-CDM and their objectives [11] ATC – Air Traffic Control, CFMU – Central Flow Management Unit

Airport CDM is now implemented in 35 European airports, the list could be found on www.euro-cdm.org/airports.php.

As an example, for Paris Charles de Gaulle airport, where A-CDM has been implemented since 2010, in 2013 departure times were respected more than 85% of the cases, against 80%

before, the ground traffic became more fluid (taxiing times have been shortened by 2 to 4 minutes), daily consumption of fuel reduced in 14.5 tons [12].

Information sharing is the most powerful concept element in A-CDM. Clearly, there is a need of proper managing of all that information. This could be done by System Wide Information Management or SWIM – an enabler for A-CDM, which will be discussed in one of the next sections.

2.4.1 A-CDM Milestones Approach

One of the essential procedure characteristics defined by A-CDM is the utilization of so called "Milestone Approach", which involves breaking the global process sequence down into milestones.

Milestones are used to mark specific points along turnaround timeline, starting from the planning of the inbound flight from the outstation until the take-off of the flight at the subject airport. The achievement or not of the individual milestones is shared and distributed among CDM partners, allowing them to identify possible deviations from schedule and appropriately respond to them. When milestone events do not appear as planned, prompting mechanism raises alarm, which triggers the reaction of the responsible partner to provide the re-planning of the operation. To give an example of how the Milestones Approach may work, imagine a situation in which boarding has not started 20 minutes before estimated departure time. In this case, the milestone event indicating "start boarding" does not appear as planned, triggering an alarm to inform the involved partners about the missed target. Subsequently, Ground Handlers may be prompted to confirm the expected duration of the delay, resulting in an updated set of milestones for the remainder of the process. As a result, all partners will be informed immediately about the late boarding, enabling them to appropriately respond to the current situation [13].

2.5 System Wide Information Management (SWIM)

Up to now, the management of information used by different members of the Aviation Community has been evolving independently. As a result of this approach, today's Air Traffic Management (ATM) information systems are insufficiently integrated, resulting in difficulties in on-time use of information.

In 1997 Eurocontrol presented the System Wide Information Management (SWIM) concept to the Federal Aviation Administration, where it has been under development ever since. SWIM is the infrastructure that will allow facilitating greater sharing of ATM system information, such as airport operational status, weather information, flight data, status of special use airspace and National Airspace System restrictions.

System Wide Information Management is essential for providing the most efficient use of airspace, managing air traffic around weather, and, what is crucial for ground handling companies, increasing common situational awareness on the ground.

Operational stakeholders do not only need to exchange network operations information but they also need computer-to-computer interface. The Network Manager (NM) aims at providing consolidated interactive interfaces enabling more effective and collaborative decision making between all stakeholders. In other words, architecture of the NM Services is provided to support the interoperability requirements of the SWIM network. In order to address different customers and needs, the Network Manager's Operations services and data are available via standardized system-to-system interfaces through the NM B2B Web Services. Since it is entirely based on open standard web technologies, the installation of proprietary software on the customers side is not required. The customers targeted are mainly Air Navigation Service Providers (ANSPs), Aircraft Operators (AOs), Airports, Handling Agents and Computerized Flight Plan Service Providers (CFSPs) (Figure 2.8).

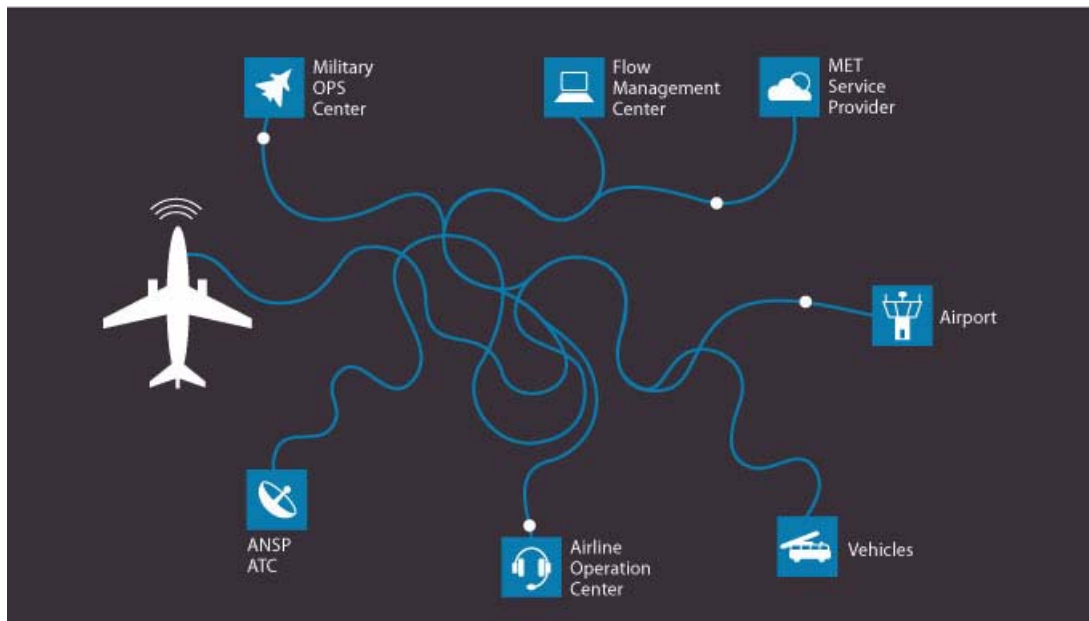


Figure 2.6: Sharing information today [14]

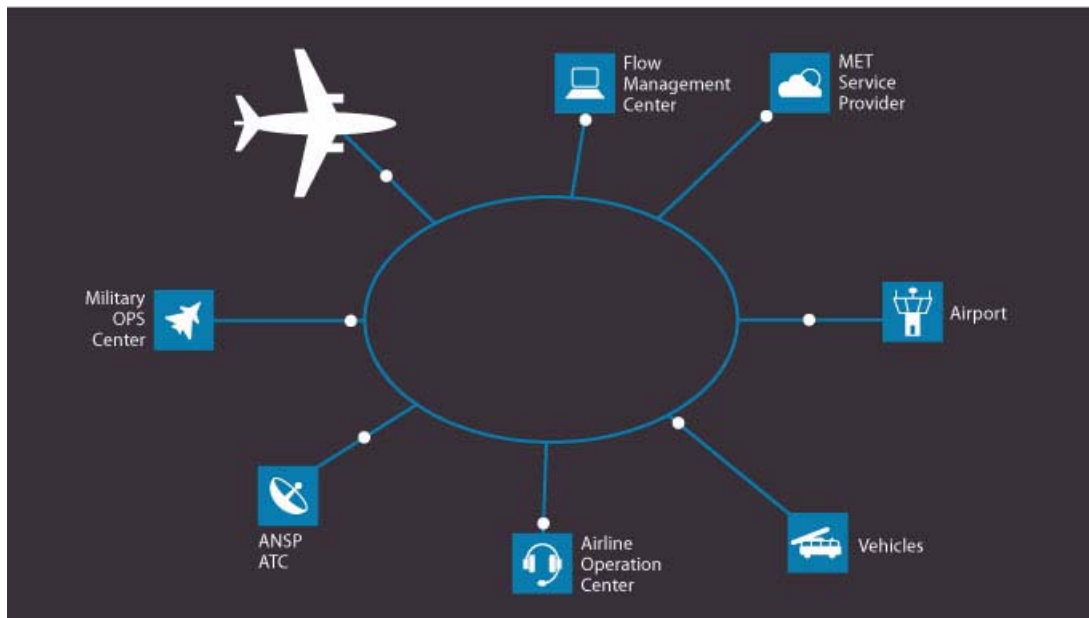


Figure 2.7: Sharing information tomorrow [14]

The NM B2B Web Services are provided in two "flavors": SOAP (Simple Object Access Protocol) web services and non-SOAP web services. The payload is always XML (Extensible Markup Language). Airspace data is predicated on the use of open data exchange standards. Within the aviation community, these are primarily Aeronautical Information Exchange Model (AIXM), Weather Information Exchange Model (WXXM) and Flight Information Exchange Model (FIXM). All services make use of HTTPS (Hypertext Transfer Protocol). Access to NM B2B Web Services requires strong authentication with digital certificates.

Up to end of 2014, 180 organizations have been entitled to access the NM B2B Web Services. More than one hundred from which access the NM B2B Web Services daily [15].

To sum up, Airport Collaborative Decision Making (A-CDM) and System Wide Information

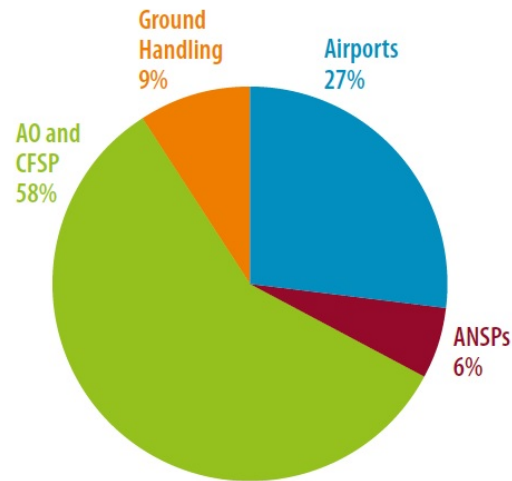


Figure 2.8: NM B2B Web Services, customer segmentation [15]

Management (SWIM) are the two modern technologies for managing and real time sharing information between aviation partners in order to improve the operational efficiency of each of them. A-CDM and SWIM are not yet implemented in every airport, but from year to year they are improved and spread up through Aviation Community.

Chapter 3

Mathematical Formulation of the Problem

3.1 Classifying Routing and Scheduling Problems

The classification of routing and scheduling problems depends on certain characteristics of the service delivery system. The simplest case is Traveling Salesman Problem (TSP), which asks the following question: Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city? The nodes may be visited in any order, the travel costs between two nodes are the same regardless of the direction traveled, there are no delivery-time restrictions and vehicle capacity is not considered. An extension of TSP, referred to as the Multiple Traveling Salesman Problem (MTSP), occurs when there are several vehicles to be routed from a single depot. If the capacity of multiple vehicles is restricted and there exists a possibility of having varying demands for each customer, the problem is classified as a Vehicle Routing Problem (VRP). If the customers being serviced have time restrictions, then the problem is referred to as Vehicle Routing Problem with Time Windows (VRPTW).

In this thesis the ground handling staff scheduling problem is presented as a Vehicle Routing Problem with Time Windows, since the aircrafts have to be serviced during time windows specified by Service Level Agreement. To simplify the model and the solution process, the complete problem is separated into multiple VRPTWs, which are identified for each worker qualification and are solved individually. Later on mathematical model will be considered for single VRPTW for specific qualification.

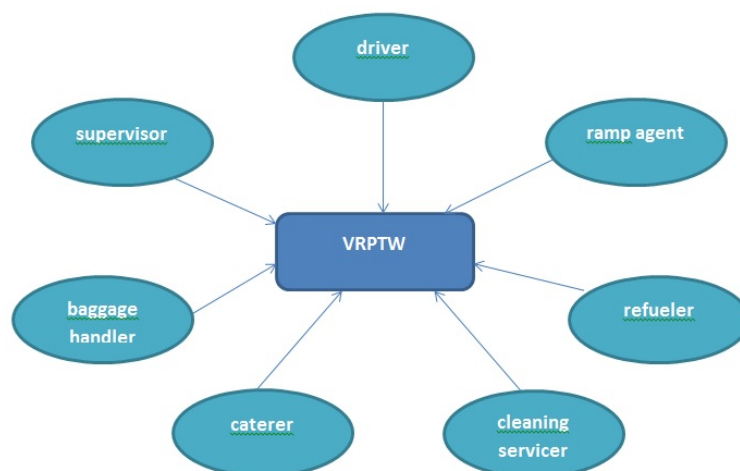


Figure 3.1

3.2 Notation

NW	number of workers
$W = \{w_1 \dots w_{NW}\}$	set of workers
NT	number of tasks
NT_i	number of assigned tasks to worker $i \in W$
$T = \{t_1 \dots t_{NT}\}$	set of tasks
$T_i = \{t_1^i \dots t_{NT_i}^i\} \subseteq T$	scheduled set of tasks performed by worker $i \in W$
NG	number of gates
$\{0\}$	depot
$G = \{g_1 \dots g_{NG}\}$	set of gates
$G_i = \{g_1^i \dots g_{NG_i}^i\} \subseteq G$	scheduled set of gates visited by worker $i \in W$
a_{ij}	time needed to move from gate $i \in G$ to gate $j \in G$
b_i	duration of shift of worker $i \in W$
d_j^i	duration of task $j \in T_i, i \in W$
$[s_j^i, f_j^i]$	scheduled time of the task $j \in T_i$ performed by worker $i \in W$
$[cs_j^i, cf_j^i]$	initial time window for task $j \in T_i, i \in W$
$[bs^i, bf^i]$	shift hours of worker $i \in W$

3.3 Input Parameters

In the proposed model it is necessary to have information from Service Level Agreements signed between airlines and ground handling companies and information about flights, workers and gates, which is the following:

- **Flights**
Timetable, which includes name of the airline, type of the aircraft, numbers of arrival and departure flights, Scheduled Time of Arrival (STA), Scheduled Time of Departure (STD) and the number of gate, where the aircraft will be serviced during its turnaround.
- **Workers**
Identification number of worker, his qualification, tasks that he is qualified to perform and his working hours.
- **Gates**
Type of the gate (finger or remote) and the time needed to move from any gate to any other.
- **Service Level Agreement (SLA)**
SLA defines strictly the order, time and duration of handling services delivered to the aircrafts.

3.4 Objective Function

In this thesis, the multi-objective approach for ground staff scheduling is introduced. Hereinafter, multi-objective approach refers to the weighted sum of performance indicators, not the Pareto based approach.

The aim is to create an optimal schedule for the workers that service aircraft at an airport during one working day. To attain that we need to minimize the following function:

$$Fitness = \alpha_1 * F_1 + \alpha_2 * F_2 + \alpha_3 * F_3 + \alpha_4 * F_4, \quad (3.1)$$

where $\alpha_1 - \alpha_4$ are weighting coefficients and $F_1 - F_4$ are objective functions.

$$F_1 = \max_{i \in W} tt_i = \max_{i \in W} \left(a_{0g_1^i} + \sum_{j=1..NT_i-1} a_{g_j^i g_{j+1}^i} + a_{g_{NT_i}^i 0} \right) \quad (3.2)$$

tt_i – total traveling time of worker $i \in W$ – is the time that worker spend on moving from gate to gate during the day. By minimizing maximum of it, we are avoiding sending the same agent successively to gates located far from each other.

$$F_2 = \max_{i \in W} wt_i = \max_{i \in W} \left(b_i - \sum_{j=1..NT_i} d_j^i - tt_i \right) \quad (3.3)$$

wt_i – total waiting time of worker $i \in W$ – is the time that worker spend between performing the tasks during the day. By minimizing maximum of it, we achieve equally distribution of the workload between the workers of the same qualifications.

$$F_3 = Penalty1 = \sum_{i \in W} \sum_{j \in T_i} 1, \quad for\ i, j \mid f_j^i < cf_j^i \quad (3.4)$$

$$F_4 = Penalty2 = \sum_{i \in W} \sum_{j \in T_i} 1, \quad for\ i, j \mid bf_i < cf_j^i \quad (3.5)$$

$Penalty1$ – is the number of planes scheduled to be serviced out of required time windows. $Penalty2$ – is the number of planes scheduled to be serviced out of workers' shift.

$(Penalty1 + Penalty2)$ is the total number of unserved aircrafts, hence by minimizing it we are leaving as few tasks uncompleted as possible.

With the aim to achieve a more flexible and accurate decision process, we are looking for a solution representing the best compromise between the four objectives listed above.

3.5 Decision Variables

Scheduled set of tasks performed by workers $T_i = \{t_1^i \dots t_{NT_i}^i\} \in T \quad \forall i \in W$ are the decision variables in our mathematical model.

3.6 Constraints

The following constraints should be respected:

- working hours of the staff

$$bs^i \leq s_j^i \leq bf^i \quad \forall j \in T_i \quad \forall i \in W \quad (3.6)$$

$$bs^i \leq s_j^i + d_j^i \leq bf^i \quad \forall j \in T_i \quad \forall i \in W \quad (3.7)$$

- time windows of the tasks

$$cs_j^i \leq s_j^i \leq cf_j^i \quad \forall j \in T_i \quad \forall i \in W \quad (3.8)$$

$$cs_j^i \leq s_j^i + d_j^i \leq cf_j^i \quad \forall j \in T_i \quad \forall i \in W \quad (3.9)$$

Time windows information is taken from Service Level Agreement (SLA).

- transportation time between the gates

$$bs_i + a_{0g_1^i} \leq s_1^i \quad \forall i \in W \quad (3.10)$$

$$f_j^i + a_{g_j^i g_{j+1}^i} \leq s_{j+1}^i \quad \forall j = 1..(NT_i - 1) \in T_i \quad \forall i \in W \quad (3.11)$$

$$f_j^i \leq bf_i - a_{g_{NT_i}^i 0} \quad \forall i \in W \quad (3.12)$$

where

$$f_j^i = s_j^i + d_j^i \quad \forall j \in T_i \quad \forall i \in W \quad (3.13)$$

Furthermore, some tasks may require several workers to cooperate. Cooperating agents have to initiate work on the task simultaneously, which also has to be maintained in the schedule.

Chapter 4

Overview of Problem Solving Methods

Combinatorial optimization problems (COP) can be regarded as a search for the best element in some set of discrete items. Determining the optimal solution of Vehicle Routing Problem with Time Windows (VRPTW) is a NP-hard problem in combinatorial optimization, hence there is no algorithm that can solve all problem instances efficiently in polynomial time. In principle, exact algorithms can be used for solving relaxed problems, with smaller amount of input or with less number of constraints. But as the size of NP-hard problem increases, the time needed to solve it grows very fast. Therefore commercial solvers tend to use heuristics to tackle real world VRPTW due to its size and the frequency that it may have to be solved.

4.1 Exact methods

Exact algorithms can range from very general techniques such as tree search with branch-and-bound to highly problem-dependent algorithms such as Dijkstra's algorithm for computing shortest path between two nodes in a weighted graph.

The exact methods for the VRPTW can be classified into three categories: Lagrange relaxation-based methods, column generation and dynamic programming.

There are several papers using slightly different approaches to Lagrange relaxation-based methods. For example, there is variable splitting approach followed by Lagrange relaxation [Fisher et al., 1997], K-tree approach followed by Lagrange relaxation [Holland, 1975], and side constraints approach followed by Lagrange relaxation [Kohl and Madsen, 1997].

The column generation approach for solving the VRPTW was used for the first time in [Desrosiers et al., 1984], and more effective version of the same model used to solve more instances to optimality was presented in [Desrosiers et al., 1992].

The dynamic programming approach for VRPTW was mentioned for the first time in [Kolen et al., 1987]. The algorithm presented in [Kohl and Madsen, 1997] uses the branch-and-bound method to achieve optimality. Branch-and-bound is based on an intelligent enumeration of the "candidate" solutions to an optimal by successively partitioning the solution space and cutting the search tree while considering limits that are calculated during the enumeration. The detailed overview of these techniques is given in [16].

We would not need any other kind of algorithm if there was not the problem that exact algorithms can take exponential time to terminate.

4.2 Heuristics

A heuristic is defined by [Reeves, 1995] as a technique which seeks good (near-optimal) solutions at a reasonable computational cost without being able to guarantee optimality, to state how close to optimality a particular feasible solution is or, in some cases, even to guarantee feasibility.

Heuristic algorithms that set routes from scratch are denoted as route-building, while algorithms that try to improve solution on the basis of already built solutions are called route-improving.

The first paper on route-building heuristics for the VRPTW is [Baker and Schaffer, 1989]. Their algorithm is an extension of the saving heuristic of the VRP [Clarke and Wright, 1964]. A similar heuristic based on the savings algorithm was developed in [Solomon, 1987] and [Landeghem, 1988]. Another heuristic described in [Solomon, 1987] is a time-oriented, nearest-neighborhood heuristic.

A problem of building one route at a time is usually that the routes generated in the latter part of the process are of poor quality as the last unvisited customers tends to be scattered over the geographic area. [Potvin and Rousseau, 1993] try to overcome this problem of the insertion heuristics by building several routes simultaneously. The initialization of the routes is done by using the insertion heuristic of [Solomon, 1987]. This method is better than the Solomon heuristics but still the solutions are quite far away from optimum. [Russell, 1995] and [Antes and Derigs, 1995] elaborate further on the insertion approach. In general, building several routes at the same time results in better solutions than building the routes one by one [17].

In general the heuristics, as f.e. the ones of [Solomon, 1987] and [Landeghem, 1988], return a solution fast. However, their solutions usually lack in quality: most of the time they differ from optimum for more than 10%.

4.3 Metaheuristics

Metaheuristics is an iterative strategy designed to find, generate, or select a heuristic that may provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information or limited computation capacity [18].

Metaheuristics do not guarantee that a globally optimal solution can be found on some class of problems. However, by searching over a large set of feasible solutions, metaheuristics can often find good solutions with less computational effort than basic heuristics and other methods. Therefore, the success of these methods is the ability to "solve in practice" some hard combinatorial problems.

There are a wide variety of metaheuristics and a number of properties along which one can characterize them.

One approach is to characterize the type of search strategy. One type of search strategy is an improvement on simple local search algorithms: Simulated Annealing, Tabu search, Iterated Local Search, Variable Neighborhood Search and Greedy Randomized Adaptive Search Procedure (GRASP). The other type of search strategy has a learning component: Ant Colony Optimization, Evolutionary Computation and Genetic Algorithms.

Metaheuristics could also be classified by single solution vs. population-based searches. Single solution approaches focus on modifying and improving a single candidate solution: Simulated Annealing, Iterated Local Search, Variable Neighborhood Search and Guided Local Search. Population-based approaches maintain and improve multiple candidate solutions, often using population characteristics to guide the search: Evolutionary Strategies, Genetic Algorithms and Particle Swarm Optimization.

4.3.1 Simulated Annealing

Simulated Annealing (SA) is the oldest metaheuristic that implemented an explicit strategy to avoid local minima. The method has its roots in statistical mechanics (Metropolis algorithm) and it was first presented as a search algorithm for combinatorial optimization problem in [19] and [20].

Simulated Annealing is inspired by the process of annealing in metallurgy, where a material is heated into a liquid state and then cooled back into a recrystallized solid state. When using Simulated Annealing, instead of searching for the best solution in the neighborhood, one simply takes solution from the neighborhood randomly. In case it is better, it is always accepted as a new current solution, but if the solution is worse - it is only accepted with a certain probability, which is determined by the gradually decreasing temperature.

The commonly used formula for acceptance probability is:

$$P(\text{accept}) = \exp\left(\frac{e - e'}{T}\right), \quad (4.1)$$

where T is the current temperature, e is the energy (or cost) of the current solution and e' is the energy of a candidate solution being considered. By reducing the temperature, the acceptance of new solution becomes more and more selective.

The size of the neighborhood considered in generating candidate solutions can change over the time or be influenced by the temperature, starting initially broad and narrowing while the algorithm is executed.

Figure 4.1 provides a pseudo code listing of the main Simulated Annealing algorithm.

```

Input: ProblemSize,  $iterations_{max}$ ,  $temp_{max}$ 
Output:  $S_{best}$ 
 $S_{current} \leftarrow \text{CreateInitialSolution}(\text{ProblemSize})$ 
 $S_{best} \leftarrow S_{current}$ 
For ( $i = 1$  To  $iterations_{max}$ )
     $S_i \leftarrow \text{CreateNeighborSolution}(S_{current})$ 
     $temp_{curr} \leftarrow \text{CalculateTemperature}(i, temp_{max})$ 
    If ( $\text{Cost}(S_i) \leq \text{Cost}(S_{current})$ )
         $S_{current} \leftarrow S_i$ 
        If ( $\text{Cost}(S_i) \leq \text{Cost}(S_{best})$ )
             $S_{best} \leftarrow S_i$ 
    End
    ElseIf ( $\text{Exp}(\frac{\text{Cost}(S_{current}) - \text{Cost}(S_i)}{temp_{curr}}) > \text{Rand}()$ )
         $S_{current} \leftarrow S_i$ 
    End
End
Return ( $S_{best}$ )

```

Figure 4.1: Pseudo code for Simulated Annealing [21]

The convergence proof suggests that the system will always converge to the global optimum if cooling period is long enough. The downside of this theoretical finding is that for some problems the number of samples taken for optimum convergence to occur can be more than a complete enumeration of the search space [21].

Restarting the cooling schedule with the best solution found so far can lead to improvements in performance for some problems. The effectiveness of the algorithm and its computation time could be improved by using a problem specific heuristic in choosing starting point for the search, by parallelization or by hybridization with other algorithms such as Tabu Search or Genetic Algorithms.

4.3.2 Tabu Search

Tabu Search basic ideas were first introduced in [22], based on earlier ideas formulated in [23]. Tabu Search explicitly uses the history of the search, both to escape from local minima and to implement an explorative strategy.

At each iteration, the neighborhood of the current solution is explored and the best solution there is selected as the new current solution. The current solution is set to the best solution in the neighborhood even if it is worse. This has to be done in order to allow the algorithm to escape from a local optimum.

To prevent cycling, it is forbidden to choose solutions that were visited recently. Latter are added to Tabu list. Often, Tabu list does not contain illegal solutions, but forbidden moves. It makes sense to allow the Tabu list to be overruled if this leads to an improvement of the current overall best solution. Criteria used for overruling the Tabu list are called aspiration criteria. Tabu Search is usually stopped after a constant number of iterations without any improvement of the overall best solution or after a constant total number of iterations.

Candidates for neighboring moves can be generated deterministically for the entire neighborhood or the neighborhood can be stochastically sampled to a fixed size, trading off efficiency for accuracy.

Figure 4.2 provides a pseudo code listing of the main Tabu Search algorithm.

```

Input:  $TabuList_{size}$ 
Output:  $S_{best}$ 
 $S_{best} \leftarrow \text{ConstructInitialSolution}()$ 
 $TabuList \leftarrow \emptyset$ 
While ( $\neg \text{StopCondition}()$ )
   $CandidateList \leftarrow \emptyset$ 
  For ( $S_{candidate} \in S_{best\_neighborhood}$ )
    If ( $\neg \text{ContainsAnyFeatures}(S_{candidate}, TabuList)$ )
       $CandidateList \leftarrow S_{candidate}$ 
    End
  End
   $S_{candidate} \leftarrow \text{LocateBestCandidate}(CandidateList)$ 
  If ( $\text{Cost}(S_{candidate}) \leq \text{Cost}(S_{best})$ )
     $S_{best} \leftarrow S_{candidate}$ 
     $TabuList \leftarrow \text{FeatureDifferences}(S_{candidate}, S_{best})$ 
    While ( $TabuList > TabuList_{size}$ )
      DeleteFeature( $TabuList$ )
    End
  End
End
Return ( $S_{best}$ )

```

Figure 4.2: Pseudo code for Tabu Search [21]

In spite of the success that Tabu Search has, there is not yet available theoretical support. No study made it possible to prove the convergence of the method. In order to improve the procedure, techniques of parallelization and combination with other approaches are proposed. The hybridization of Tabu search and Simulated Annealing was proposed in [Osman, 1993] and [Tuytens et al., 1994], and the hybridization of Tabu Search and Genetic Algorithm in [Glover et al., 1995].

4.3.3 Particle Swarm Optimization

The Particle Swarm Optimization (PSO) is a recently proposed metaheuristic method [Kennedy and Eberhart, 1995], [Eberhart et al., 2001] that simulates the collective behavior of the animal displacements, e.g. the fish school or bird flock, where the individuals themselves have access only to limited information, such as position and the speed of their closest neighbors. It can be observed that a fish school is able to avoid a predator in the following manner: initially it gets divided into two groups, then the original school is reformed (Figure 4.3), while maintaining the cohesion among the school.

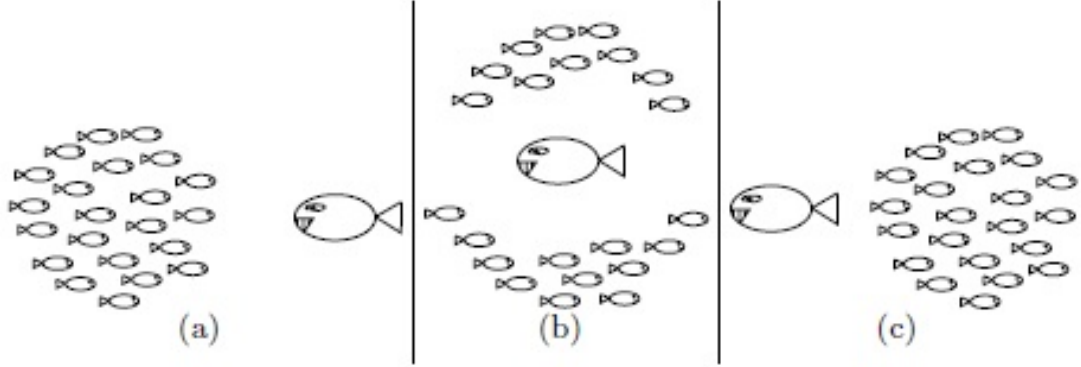


Figure 4.3: A schematic of a fish school avoiding predator. (a) the school forms only one group, (b) the individuals avoid the predator by forming a "fountain" like structure, (c) the school is reformed [24]

The standard PSO algorithm works by having a population of candidate solutions, which move around in the search space according to the particle's position and velocity. The movement of a particle is guided by its own best-known position and the entire swarm's best-known position in order to move all the particles toward the global best solution in the solution space. At each iteration, the swarm's best-known position is updated. The process is repeated for a given number of iterations.

Each iteration a particle's velocity is updated using:

$$v_i(t+1) = v_i(t) + \left(c_1 * rand() * (p_i^{best} - p_i(t)) \right) + \left(c_2 * rand() * (p_{gbest} - p_i(t)) \right), \quad (4.2)$$

where $v_i(t+1)$ is the new velocity for the i^{th} particle, c_1 and c_2 are the weighting coefficients for the personal best and global best positions respectively, $p_i(t)$ is the i^{th} particle's position at time t , p_i^{best} is the i^{th} particle's best known position, and p_{gbest} is the best position known to the swarm. The $rand()$ function generates a uniformly random variable $\in [0, 1]$.

A particle's position is updated using:

$$p_i(t+1) = p_i(t) + v_i(t). \quad (4.3)$$

Figure 4.4 provides a pseudo code listing of the standard Particle Swarm Optimization algorithm.


```

Input: ProblemSize,  $Population_{size}$ 
Output:  $P_{g.best}$ 
Population  $\leftarrow \emptyset$ 
 $P_{g.best} \leftarrow \emptyset$ 
For ( $i = 1$  To  $Population_{size}$ )
     $P_{velocity} \leftarrow \text{RandomVelocity}()$ 
     $P_{position} \leftarrow \text{RandomPosition}(Population_{size})$ 
     $P_{p.best} \leftarrow P_{position}$ 
    If ( $\text{Cost}(P_{p.best}) \leq \text{Cost}(P_{g.best})$ )
         $P_{g.best} \leftarrow P_{p.best}$ 
    End
End
While ( $\neg \text{StopCondition}()$ )
    For ( $P \in \text{Population}$ )
         $P_{velocity} \leftarrow \text{UpdateVelocity}(P_{velocity}, P_{g.best}, P_{p.best})$ 
         $P_{position} \leftarrow \text{UpdatePosition}(P_{position}, P_{velocity})$ 
        If ( $\text{Cost}(P_{position}) \leq \text{Cost}(P_{p.best})$ )
             $P_{p.best} \leftarrow P_{position}$ 
            If ( $\text{Cost}(P_{p.best}) \leq \text{Cost}(P_{g.best})$ )
                 $P_{g.best} \leftarrow P_{p.best}$ 
            End
        End
    End
End
Return ( $P_{g.best}$ )

```

Figure 4.4: Pseudo code for Particle Swarm Optimization [21]

The number of particles should be low, around 20-40. The speed with which a particle can move (maximum change in its position per iteration) should be bounded, such as to a percentage of the size of the domain. Particles may leave the boundary of the problem space and may be penalized, be reflected back into the domain or biased to return back toward a position in the problem domain. An inertia or momentum coefficient can be introduced to limit the change in velocity.

4.3.4 Ant Colony Optimization

The Ant Colony Optimization (ACO) algorithms are inspired by the foraging behavior of ants. While walking from food sources to the nest and vice versa, ants deposit pheromone on the ground. When they decide about a direction to go, with higher probability they choose paths that are marked by stronger pheromone concentrations. This basic behavior is the basis for a cooperative interaction which leads to the emergence of shortest paths.

Ant Colony System is the most popular ACO algorithm. Its pseudo code listing is provided on Figure 4.5.

The probabilistic step-wise construction of solution makes use of both history (pheromone) and problem-specific heuristic information to construct a solution piece-by-piece. Each component can only be selected if it has not already been chosen. For components that can be selected given current component i , the selection probability is defined as:

$$P_{i,j} \leftarrow \frac{\tau_{i,j}^\alpha * \eta_{i,j}^\beta}{\sum_{k=1}^c \tau_{i,k}^\alpha * \eta_{i,k}^\beta}, \quad (4.4)$$

where $\eta_{i,j}$ is the maximizing contribution to the overall score of selecting the component, β is the heuristic coefficient (commonly fixed to 1.0), $\tau_{i,j}$ is the pheromone value for the component,

α is the history coefficient, and c is the set of usable components. Greediness factor q^0 is used to decide when to use the above probabilistic component selection and when to greedily select the best possible component.

A local pheromone update is performed for each solution that is constructed to dissuade following solutions to use the same components in the same order, as follows:

$$\tau_{i,j} \leftarrow (1 - \sigma) * \tau_{i,j} + \sigma * \tau_{i,j}^0, \quad (4.5)$$

where $\tau_{i,j}$ represents the pheromone for the component (i, j) , σ is the local pheromone factor and $\tau_{i,j}^0$ is the initial pheromone value.

At the end of each iteration, the pheromone is updated and decayed using the best candidate solution found so far as follows:

$$\tau_{i,j} \leftarrow (1 - \rho) * \tau_{i,j} + \rho * \Delta\tau_{i,j}, \quad (4.6)$$

where $\tau_{i,j}$ represents the pheromone for the component (i, j) , ρ is the decay factor, and $\Delta\tau_{i,j}$ is the maximizing solution cost for the best solution found so far if the component ij is used in the globally best known solution, otherwise it is 0.

```

Input: ProblemSize, Populationsize,  $m, \rho, \beta, \sigma, q^0$ 
Output:  $P_{best}$ 
 $P_{best} \leftarrow \text{CreateHeuristicSolution}(\text{ProblemSize})$ 
 $P_{best\_cost} \leftarrow \text{Cost}(S_h)$ 
 $\text{Pheromone}_{init} \leftarrow \frac{1.0}{\text{ProblemSize} \times P_{best\_cost}}$ 
 $\text{Pheromone} \leftarrow \text{InitializePheromone}(\text{Pheromone}_{init})$ 
While ( $\neg \text{StopCondition}()$ )
  For ( $i = 1$  To  $m$ )
     $S_i \leftarrow \text{ConstructSolution}(\text{Pheromone}, \text{ProblemSize}, \beta, q^0)$ 
     $S_{i\_cost} \leftarrow \text{Cost}(S_i)$ 
    If ( $S_{i\_cost} \leq P_{best\_cost}$ )
       $P_{best\_cost} \leftarrow S_{i\_cost}$ 
       $P_{best} \leftarrow S_i$ 
    End
     $\text{LocalUpdateAndDecayPheromone}(\text{Pheromone}, S_i, S_{i\_cost}, \sigma)$ 
  End
   $\text{GlobalUpdateAndDecayPheromone}(\text{Pheromone}, P_{best}, P_{best\_cost}, \rho)$ 
End
Return ( $P_{best}$ )

```

Figure 4.5: Pseudo code for Ant Colony System [21]

The local pheromone coefficient σ controls the amount of contribution history plays in a components probability of selection and is commonly set to 0.1. The heuristic coefficient β controls the amount of contribution problem-specific heuristic information plays in a components probability of selection and is commonly chosen between 2 and 5. The decay factor ρ controls the rate at which historic information is lost and is commonly set to 0.1. The greediness factor q^0 is commonly set to 0.9. The total number of ants m is commonly set low, such as 10 [21].

4.3.5 Evolutionary Algorithms

Contrary to Simulated Annealing and Tabu Search, Evolutionary Algorithms operate not only in one solution, but in a population of solutions. The population is arbitrarily initialized, and it evolves toward better regions of the search space by means of randomized processes of selection, mutation and recombination. One iteration of this loop is referred to as a generation. Each individual in the population receives a measure of its fitness in the environment. Selection focuses attention on high fitness individuals, thus exploiting the available fitness information. Mutation introduces innovation into the population and recombination mechanism allows mixing of parental information while passing it to their descendants.

Three main streams of evolutionary algorithms can be identified [25]:

- Genetic Algorithms (GA), developed by J. Holland [Holland, 1975] in the United States of America, with refinements by K. De Jong [De Jong, 1975], J. Grefenstette [Grefenstette, 1986], [Grefenstette, 1987] and D. Goldberg [Goldberg, 1989]
- Evolutionary Programming (EP), originally developed by L. J. Fogel, A. J. Owens, and M. J. Walsh in the United States of America [Fogel, Owens and Walsh, 1966] and recently refined by D. B. Fogel [Fogel, 1991b]
- Evolution Strategies (ES), developed in Germany by I. Rechenberg [Rechenberg 1965; Rechenberg 1973] and H.-P. Schwefel [Schwefel, 1977], [Schwefel, 1981]

Although simplistic from a biologist's point of view, these algorithms are sufficiently complex to provide robust and powerful adaptive search mechanisms.

4.3.5.1 Genetic Algorithm

The Genetic Algorithm is inspired by population genetics and evolution at the population level. Individuals of a population contribute their genotypes (genetic material) proportional to the suitability to the environment of their phenotype (expressed genome), in the form of offspring. The next generation is created through mating that involves recombination of two individuals' genomes with the introduction of mutation (random copying errors). This iterative process may result in an improved adaptive-fit between the phenotypes of individuals in a population and the environment.

Genetic Algorithm is implemented as following:

1. Randomly generate the first population.
2. Do until the termination criterion is satisfied:
 - (a) Evaluate the fitness of all of the individuals in the population.
 - (b) Create a new population by performing operations such as crossover, fitness-proportionate reproduction and mutation on the individuals whose fitness has just been measured.
 - (c) Discard the old population and iterate using the new population.

The pseudo code listing of Genetic Algorithm is provided on Figure 4.6.

```

Input:  $Population_{size}, Problem_{size}, P_{crossover}, P_{mutation}$ 
Output:  $S_{best}$ 
Population  $\leftarrow$  InitializePopulation( $Population_{size}, Problem_{size}$ )
EvaluatePopulation(Population)
 $S_{best} \leftarrow$  GetBestSolution(Population)
While ( $\neg$ StopCondition())
    Parents  $\leftarrow$  SelectParents(Population,  $Population_{size}$ )
    Children  $\leftarrow \emptyset$ 
    For ( $Parent_1, Parent_2 \in$  Parents)
         $Child_1, Child_2 \leftarrow$  Crossover( $Parent_1, Parent_2, P_{crossover}$ )
        Children  $\leftarrow$  Mutate( $Child_1, P_{mutation}$ )
        Children  $\leftarrow$  Mutate( $Child_2, P_{mutation}$ )
    End
    EvaluatePopulation(Children)
     $S_{best} \leftarrow$  GetBestSolution(Children)
    Population  $\leftarrow$  Replace(Population, Children)
End
Return ( $S_{best}$ )

```

Figure 4.6: Pseudo code for Genetic Algorithm [21]

The size of the population must be large enough to provide sufficient coverage of the domain and mixing of the useful sub-components of the solution [Goldberg, 1992]. The Genetic Algorithm is classically configured with a high probability of recombination (95%-99% of the selected population) and a low probability of mutation ($1/L$, where L is the number of components in a solution) [Muhlenbein, 1992] [Back, 1993]. The fitness-proportionate selection of candidate solutions to contribute to the next generation should be neither too greedy (to avoid the takeover of fitter candidate solutions) nor too random.

4.3.5.2 Evolutionary Programming

Evolutionary Programming, as well as Evolution Strategies described below, is inspired by the theory of evolution by means of natural selection. Specifically, the technique is inspired by the species-level process of evolution (phenotype, hereditary, variation) and is not concerned with the genetic mechanisms of evolution (genome, chromosomes, genes).

Evolutionary Programming is implemented as following:

1. Choose an initial population of trial solutions at random.
2. Replicate each solution into a new population.
3. Mutate each of these offspring solutions according to a distribution of mutation types.
4. Assess each offspring solution by computing its fitness.

The pseudo code listing of Evolutionary Programming is provided on Figure 4.7.

```
Input:  $Population_{size}$ , ProblemSize, BoutSize
Output:  $S_{best}$ 
Population  $\leftarrow$  InitializePopulation( $Population_{size}$ , ProblemSize)
EvaluatePopulation(Population)
 $S_{best} \leftarrow$  GetBestSolution(Population)
While ( $\neg$ StopCondition())
    Children  $\leftarrow \emptyset$ 
    For ( $Parent_i \in$  Population)
         $Child_i \leftarrow$  Mutate( $Parent_i$ )
        Children  $\leftarrow Child_i$ 
    End
    EvaluatePopulation(Children)
     $S_{best} \leftarrow$  GetBestSolution(Children,  $S_{best}$ )
    Union  $\leftarrow$  Population + Children
    For ( $S_i \in$  Union)
        For (1 To BoutSize)
             $S_j \leftarrow$  RandomSelection(Union)
            If ( $Cost(S_i) < Cost(S_j)$ )
                 $Si_{wins} \leftarrow Si_{wins} + 1$ 
            End
        End
    End
    Population  $\leftarrow$  SelectBestByWins(Union,  $Population_{size}$ )
End
Return ( $S_{best}$ )
```

Figure 4.7: Pseudo code for Evolutionary Programming [21]

The sample size (BoutSize) for tournament selection during competition is commonly chosen between 5% and 10% of the population size. There is no requirement neither that the population size has to be held constant, nor that only a single offspring can be generated from each parent. Typically, Evolutionary Programming does not use any crossover as a genetic operator and uses stochastic tournament to determine solutions to be chosen for the next population.

4.3.5.3 Evolution Strategies

Schwefel generalized evolution strategies into two types: plus-strategy, denoted by $(\mu + \lambda)$ and comma-strategy, denoted by (μ, λ) [26]. Both of them imitate the following basic principle: a population, leading to the possibility of recombination with random mating, mutation and selection. In the plus case, the parental generation is taken into account during selection, while in the comma case only the offspring undergoes selection, and the parents die off. μ denotes the population size, and λ denotes the number of offspring generated per generation.

The pseudo code listing of (μ, λ) Evolution Strategy is provided on Figure 4.8.

```

Input:  $\mu, \lambda, \text{ProblemSize}$ 
Output:  $S_{best}$ 
Population  $\leftarrow$  InitializePopulation( $\mu, \text{ProblemSize}$ )
EvaluatePopulation(Population)
 $S_{best} \leftarrow$  GetBest(Population, 1)
While ( $\neg \text{StopCondition}()$ )
    Children  $\leftarrow \emptyset$ 
    For ( $i = 0$  To  $\lambda$ )
         $Parent_i \leftarrow$  GetParent(Population,  $i$ )
         $S_i \leftarrow \emptyset$ 
         $S_{i\_problem} \leftarrow$  Mutate( $P_{i\_problem}^i, P_{i\_strategy}^i$ )
         $S_{i\_strategy} \leftarrow$  Mutate( $P_{i\_strategy}^i$ )
        Children  $\leftarrow S_i$ 
    End
    EvaluatePopulation(Children)
     $S_{best} \leftarrow$  GetBest(Children +  $S_{best}$ , 1)
    Population  $\leftarrow$  SelectBest(Population, Children,  $\mu$ )
End
Return ( $S_{best}$ )

```

Figure 4.8: Pseudo code for (μ, λ) Evolution Strategy [21]

Evolution Strategies are commonly configured such that $1 \leq \mu \leq \lambda$. The ratio of μ to λ influences the amount of selection pressure (greediness) exerted by the algorithm. The comma-selection variation of the algorithm can be good for dynamic problem instances given its capability for continued exploration of the search space, whereas the plus-selection variation can be good for refinement and convergence.

Each of considered algorithms emphasizes different features as being most important for a successful evolution process.

Both ES and EP are concentrated on mutation as the main search operator, while the role of mutation in canonical GA is usually seen to be of secondary (if any) importance. On the other hand, recombination that plays a major role in canonical GA, is missing completely in EP, and is urgently necessary for use in connection to self-adaptation in ES.

Finally, both canonical GA and EP emphasize a necessarily probabilistic selection mechanism, while from the ES point of view selection is completely deterministic without any evidence for the necessity of incorporating probabilistic rules. In contrast, both ES and EP definitely exclude some individuals from being selected for reproduction, i.e., they use extinctive selection mechanisms, while canonical GAs generally assign a nonzero selection probability to each individual.

In current thesis, Evolutionary Algorithms and, specifically, Evolution Strategies were chosen for solving Vehicle Routing Problem with Time Windows. There are several reasons for that decision. The main reason is that Evolutionary Algorithms operate on the big search space; they are working not only with one solution, but with a population of solutions. Furthermore it is possible to introduce changes in formulation of the problem, mute on new parameters into the algorithm and add new constraints. The last but not least reason is the possibility of parallel computing with the usage of clusters.

Chapter 5

Decision Support Tool

5.1 Algorithm Outline

As it was mentioned before, in this thesis ground handling staff scheduling problem is presented as multiple Vehicle Routing Problems with Time Windows (VRPTW), identified for each worker qualification.

VRPTW is a combinatorial optimization and integer programming problem, the solution of which is the optimal set of routes for a fleet of vehicles to traverse in order to deliver services to a given set of customers within the specified time windows.

Ground handling agents are considered as vehicles; tasks that need to be done during aircrafts' turnaround are considered as customers.

Evolutionary Strategies were chosen to look for a solution of VRPTW. The reasons for this choice were given in the previous chapter.

Basic steps of ES algorithm are the following:

1. Initialize individuals (candidate solutions) to form the initial parent population
2. Repeat until termination criterion is satisfied:
 - (a) Select mating pool from the parent population to generate offsprings
 - (b) Mutate children
 - (c) Select the best individuals from the population of children and parents based on the values of fitness function
 - (d) Use the selected individuals as parent population for the next generation.

In Evolutionary Strategies, population is the list of individuals, which are represented by their phenotype, containing the candidate values of the parameters being optimized. Therefore in VRPTW for ground handling, individuals are the possible set of routes, which consist of sequences of customers for each vehicle.

Two case studies corresponding to two different implementations of algorithm will be considered in this thesis. The main difference of case study 2 compared to case study 1 is that all the individuals generated initially or obtained as a result of applying genetic operators represent feasible solutions, i. e. they satisfy all constraints described in section 3.6.

5.2 Initial Population

5.2.1 Case 1

The first population is generated randomly. This means, that which customers, how many of them and in which order they have to be visited by each vehicle are chosen at random, while making sure that all the customers are visited exactly one time. This procedure has to be done for each individual in first population.

It is also essential to check that worker is able to perform all tasks assigned to him. As for example, pushback operation has to be done by a driver, though it cannot be assigned to the driver that can only perform bus driving.

5.2.2 Case 2

The initial population is no longer generated randomly (*generateInitialPopulation(...)*, Annex A). The route for the first vehicle is being created in the following way: the customer is added to the end of the current route if and only if both time windows constraints and working hours constraints as well as the ability to perform this particular task are satisfied (*CheckTAS(...)*, Annex A).. This procedure is repeated till there are no more customers that could be added to the route. Then the same idea is applied to construct routes for other vehicles, though only unvisited yet customers are considered. Each individual in population is generated in described way. In this case the feasibility of initial solutions is kept.

5.3 Termination Criterion

In this work, as a termination criterion was chosen the number of iterations. The another idea would be to use the difference in resulting values of fitness function. However, in that case we would need to introduce changes in termination criterion each time when tuning weights in fitness function, which will result in difficulties in comparison different runs of the algorithm.

5.4 Tournament Selection

Mating pool is of size of population and is created using Ten-Tournament selection (*CreateMatingPool(...)*, Annex A). This type of selection involves running several tournaments among ten individuals chosen at random from the population. The winner of each tournament (the one with the best fitness) is selected for mating pool (*Tournament(...)*, Annex A).

Child is equal to its parent, randomly chosen from mating pool, modified with some probability by genetic operations (mutated).

The new generation is made out of children and parents using elitism – individuals are sorted by fitness, after that the best half is chosen.

5.5 Genetic Operations

5.5.1 Case 1

In this work children could undergo two mutations with different probabilities (*GeneticOperators(...)*, Annex A). Those mutations were chosen in order to maintain the genetic diversity from one generation of population to the next one while complying with the rule that all the customers have to be visited exactly once.

- **Mutation 1**

Changing the order of visiting customers for the same vehicle, f. e.:

route1: [2 9 10]	→	route1: [9 2 10]
route2: [4 6]	→	route2: [6 4]
route3: [3 1]	→	route3: [3 1]
route4: [5 7 8]	→	route4: [8 7 5]

Individual can undergo this mutation for several vehicles.

- **Mutation 2**

Swapping of parts of the routes between two different vehicles, f. e.:

route1: [9 2 10]	→	route1: [3 1 10]
route2: [6 4]	→	route2: [6 4]
route3: [3 1]	→	route3: [9 2]
route4: [8 7 5]	→	route4: [8 7 5]

As mutation 1, individual can undergo mutation 2 several times. The additional benefit of this mutation is that part of the first route could be swapped with a zero part of the second route (or the other way around), therefore mutation 2 will work as an insertion of part of one route into another.

After the mutations, heuristic was added to speed up the convergence. The idea of heuristic is to find the shortest non-zero route and add it in the end of other randomly chosen route. This "merging" helps to minimize the number of situations when a vehicle has to serve just 1-2 customers during the whole day.

5.5.2 Case 2

Here genetic operation replicates mutation 2 from the previous case with the difference that mutation is only applied if it will not violate the feasibility.

5.6 Fitness Function

5.6.1 Case 1

Fitness function is the objective function described in the previous Chapter (*CalculateFitness(...)*, Annex A). Total waiting time and total traveling time are computed with the use of function (*WaitTravelTime(...)*, Annex A). Total number of unserved planes are computed by (*Penalty(...)*, Annex A).

5.6.2 Case 2

Since individuals in population satisfy a priori all necessary constraints, fitness function is only introduced as the weighted sum of total travelling time and total waiting time.

Above were presented the most significant properties of two implementations of the Evolution Strategies for VRPTW, considered in this thesis.

Chapter 6

Case Studies

All the experiments described in this chapter were conducted on the computer with the following properties:

Processor	Intel(R) Core(TM) i5-3317U CPU @ 1.70GHz
RAM	8 GB
System	64-bit operating system, processor x64

6.1 Input Data

Semi realistic data from Barcelona El Prat Airport was used to test performance of the algorithm in this thesis. The algorithm was implemented in Matlab, which is a great environment to quickly prototype scientific algorithms with the benefit of being able to set a breakpoint and then run preceding the breakpoint code and plot data for interim results. However execution time is slow compared to almost every other programming language. That is the reason why only partial data was used to imitate the real situation. The input consists of 4 gates, 50 flights of 2 airlines with 2 different types of aircrafts which are scheduled from 6am till midnight and 10-12 workers of each qualification. The input details are described below.

- **Gates**

$G = \{g_1 \dots g_4\}$ – set of gates

g_1 and g_2 are of the type finger; gates g_3 and g_4 are of the type remote; The difference between those two types of gates is that in cases when the aircraft is serviced on a remote gate, there should be provided bus services for disembarking/boarding of passengers.

$$D(G) = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 0 & 1 & 2 & 2 \\ 2 & 1 & 0 & 1 & 3 \\ 3 & 2 & 1 & 0 & 4 \\ 4 & 2 & 3 & 4 & 0 \end{bmatrix} \end{matrix}$$

$D(G)$ – cost distance matrix between the gates. The entry $(D(G))_{ij}$ is the time in minutes spent by worker to move from the gate g_i to g_j , $i, j = 1..5$.

Gate g_5 corresponds to the depot $\{0\}$.

It is considered that gates are located in line and on the same distance from each other; therefore workers need 1 minute to move from gate 1 to 2 or 2 to 3, but 2 minutes to move from gate 1 to 3 and so on.

- **Flights**

Flight timetable is presented in the following way:

Airline	Aircraft	Flight	STA	Flight	STD	Gate
VY	A330-200	3009	00615	3011	00700	3
LH	A340-300	5035	00635	5036	00735	1
VY	A330-200	200	00640	202	00725	2
LH	A340-300	1269	00700	1270	00800	4
VY	A330-200	3165	00715	3163	00800	3
LH	A340-300	6035	00740	6036	00840	2
VY	A330-200	1072	00815	1071	00900	1
VY	A330-200	214	00820	215	00905	3
VY	A330-200	5002	00820	5003	00905	4
...						

In the table above STA stands for Scheduled Time of Arrival, which is presented in a way "DHHMM", where D is a day, HH refers to hours (24 hour time format) and MM denotes minutes. STD stands for Scheduled Time of Departure and is also of type "DHHMM".

It is considered that turnaround time of aircrafts A330-200 operated by Vueling takes 45 min, while for A340-300 operated by Lufthansa it is 60 min.

- **Workers**

Working hours of ground handling agents are represented as following table:

Name	Qualification	Tasks	Start	End
Juan	supervisor	supervising	00600	01500
Roger	supervisor	supervising	01200	02100
Alberto	driver	bus, pushback	00600	01500
Manuel	driver	bus	01200	02100
Christian	ramp agent	disembarking, boarding	02100	10600
Andreas	refueler	refueling	00600	01500
Olivier	baggage handler	unloading, loading	01200	02100
Alexey	caterer	catering	02100	10600
Vladimir	cleaning servicer	cleaning	00600	01500
...				

To each worker one of the following shifts is assigned:

Morning shift	06.00 - 15.00
Afternoon shift	12.00 - 21.00
Night shift	21.00 - 06.00

Workers of the same qualification could be able to carry out different tasks. For example, driver Alberto can perform both bus driving and pushback operations, while driver Manuel is only able to do the bus driving.

- **Tasks**

The list of tasks, their duration and qualification of workers needed to perform these tasks are determined for each type of aircraft separately. For example, for A330-200 this information is performed in the following way:

Task	Qualification	Duration
supervising	supervisor	60
bus service for unboarding of passengers	driver	18
bus service for boarding of passengers	driver	20
pushback	driver	2
disembarking of passengers	ramp agent	8
boarding of passengers	ramp agent	15
refueling	refueler	13
unloading of luggage	baggage handler	11
loading of luggage	baggage handler	18
catering	caterer	15
cleaning	cleaning servicer	13

Duration of the tasks is given in minutes.

- **Service Level Agreement (SLA)**

Necessary information from SLA is given in the form:

Airline	Aircraft	Task	Number of workers	Duration	Trigger start	Time	Trigger end	Time
LH	A340-300	cleaning	1	18	STA	12	STD	-20
LH	A340-300	catering	1	20	STA	12	STD	-20
LH	A340-300	unloading	1	22	STA	0	STA	25
LH	A340-300	loading	1	26	STA	25	STD	0
LH	A340-300	pushback	1	2	STD	0	STD	2
VY	A330-200	supervising	1	60	STA	-5	STD	10
VY	A330-200	bus	2	18	STA	-5	STA	13
VY	A330-200	bus	2	20	STD	-20	STD	0
VY	A330-200	disembarking	1	8	STA	0	STA	8
VY	A330-200	boarding	1	15	STD	-15	STD	0
...								

Values in columns "Duration" and "Time" are given in minutes.

In this thesis, the following precedence restrictions are represented in the schedule through Service Level Agreement table:

- Loading of baggage could be performed only after unloading.
- Fueling, catering and servicing could be performed only in between disembarking and boarding of passengers.
- Bus service has to be performed during the whole process of disembarking of passengers plus some time before/after, and during the whole process of boarding of passengers plus some time before.

There are also some parameters that have to be tuned before running the code: size of population, maximum number of iterations, probability of choosing the best individual in tournament selection, probabilities of mutations 1 and 2, as well as weights used in fitness function.

6.2 Case Study 1

6.2.1 Presentation of Results

To get the full schedule for ground handling agents Matlab code should be rerun for each qualification of the workers. There are slight differences between formulations of VRPTW for some of workers' qualifications that were taken into account in programming code in this thesis. For example, the fact that some of workers are able to perform more than one task, or the fact that aircrafts need to be serviced by workers of some qualification twice (ramp agents performing boarding and unboarding of passengers, bus drivers, etc.)

Performing and analysing the full schedule is a tedious task, therefore below will be presented results achieved by running the algorithm for one of the qualifications: refuelers.

The algorithm was run with the following parameters:

$PSize = 500$	population size
$MaxIter = 1000$	number of iterations
$\alpha_1 = 4$	weight for travel time in fitness function
$\alpha_2 = 2$	weight for waiting time in fitness function
$\alpha_3 = 100$	weight for number of planes serviced out of time windows in fitness function
$\alpha_4 = 10000$	weight for number of planes serviced out of shift time in fitness function
$pMutation1 = 0.3$	probability of mutation1
$pMutation2 = 0.4$	probability of mutation2

for 8 workers of a qualification "refueler", which could only perform one task "refueling" and whose working hours are the following:

Name	Qualification	Tasks	Start	End
Juan	refueler	refueling	00600	01500
Alberto	refueler	refueling	00600	01500
Andreas	refueler	refueling	01200	02100
Christian	refueler	refueling	01200	02100
Olivier	refueler	refueling	01200	02100
Alexey	refueler	refueling	01200	02100
Vladimir	refueler	refueling	01900	10100
Rodrigo	refueler	refueling	01900	10100

The output of the model is performed as a schedule and is shown on Table 6.1:

Name	Qualification	Tasks	Start	End	Flight	Gate
Juan	refueler	refueling	0:06:23	0:06:36	3009	3
Juan	refueler	refueling	0:06:48	0:07:01	200	2
Juan	refueler	refueling	0:07:12	0:07:30	1269	4
Juan	refueler	refueling	0:10:32	0:10:50	1036	2
Juan	refueler	refueling	0:11:48	0:12:01	105	2
Juan	refueler	refueling	0:13:18	0:13:31	420	1
Alberto	refueler	refueling	0:06:47	0:07:05	5035	1
Alberto	refueler	refueling	0:10:57	0:11:15	2098	1
Alberto	refueler	refueling	0:11:17	0:11:35	100	3
Alberto	refueler	refueling	0:11:42	0:12:00	1038	4
Alberto	refueler	refueling	0:13:57	0:14:15	657	2
Andreas	refueler	refueling	0:15:12	0:15:30	689	1
Andreas	refueler	refueling	0:15:33	0:15:46	5002	4
Andreas	refueler	refueling	0:16:12	0:16:30	3400	3
Andreas	refueler	refueling	0:16:42	0:17:00	345	2
Andreas	refueler	refueling	0:18:08	0:18:21	1277	3
Andreas	refueler	refueling	0:18:42	0:19:00	7000	4
Christian	refueler	refueling	0:13:48	0:14:01	288	3
Christian	refueler	refueling	0:15:08	0:15:21	4000	3
Christian	refueler	refueling	0:15:21	0:15:34	3165	3
Christian	refueler	refueling	0:16:12	0:16:30	266	4
Christian	refueler	refueling	0:17:47	0:18:05	540	1
Christian	refueler	refueling	0:20:08	0:20:21	990	1
Christian	refueler	refueling	0:20:33	0:20:46	134	3
Olivier	refueler	refueling	0:12:08	0:12:21	7084	1
Olivier	refueler	refueling	0:12:23	0:12:36	214	3
Olivier	refueler	refueling	0:12:48	0:13:01	354	2
Olivier	refueler	refueling	0:15:08	0:15:21	5544	4
Olivier	refueler	refueling	0:19:23	0:19:36	4067	3
Olivier	refueler	refueling	0:20:03	0:20:16	9226	4
Olivier	refueler	refueling	0:20:17	0:20:35	1266	3
Alexey	refueler	refueling	0:12:01	0:12:14	1072	1
Alexey	refueler	refueling	0:13:48	0:14:01	5403	4
Alexey	refueler	refueling	0:16:27	0:16:45	1081	1
Alexey	refueler	refueling	0:17:53	0:18:06	1021	2
Alexey	refueler	refueling	0:18:07	0:18:25	1034	1
Alexey	refueler	refueling	0:18:58	0:19:11	1002	1
Vladimir	refueler	refueling	0:19:02	0:19:15	698	2
Vladimir	refueler	refueling	0:19:17	0:19:35	5039	4
Vladimir	refueler	refueling	0:20:07	0:20:25	1356	2
Vladimir	refueler	refueling	0:21:38	0:21:51	3008	4
Vladimir	refueler	refueling	0:21:53	0:22:06	190	2
Rodrigo	refueler	refueling	0:21:13	0:21:26	2587	1
Rodrigo	refueler	refueling	0:21:27	0:21:40	2087	2
Rodrigo	refueler	refueling	0:21:40	0:21:53	257	2
Rodrigo	refueler	refueling	0:22:18	0:22:31	1012	1
Rodrigo	refueler	refueling	0:22:42	0:23:00	3210	2
Rodrigo	refueler	refueling	0:23:01	0:23:19	1044	3
Rodrigo	refueler	refueling	0:23:23	0:23:36	344	1
Rodrigo	refueler	refueling	0:23:37	0:23:55	6035	2

Table 6.1: Output of the algorithm. Served in time aircrafts are highlighted in green, unserved aircrafts are highlighted in red

As it could be seen from Table 6.1 algorithm does not find a feasible solution: 11 aircrafts out of 50 stay unserved. If the number of iterations and population size is increased (as f. e. till 5000), the number of unserved aircrafts is smaller but even then the algorithm cannot find the feasible solution. In case of big number of iterations and population size, the computational time will increase to many hours, which will kill the utility of algorithm in real life.

Figure 6.1 shows tendency of fitness function of best individual in population on each iteration. The value of fitness function decreases but in much slower tempo when increasing the number of iterations.

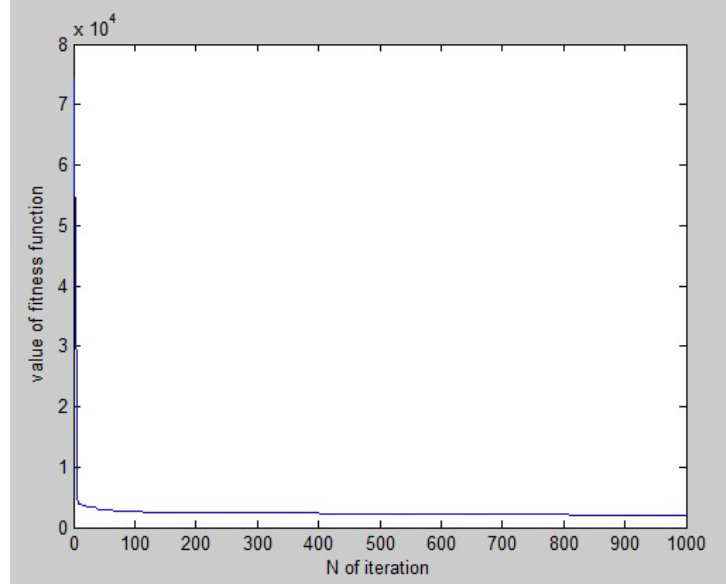


Figure 6.1: Fitness function of best individuals on each population

From Figure 6.2 and Figure 6.3 could be seen that the algorithm immediately find solutions with zero *Penalty1* - number of planes scheduled to be serviced out of shift time, but looking for solutions with zero *Penalty2* - number of planes scheduled to be serviced out of required time windows goes much slower. That make sense since at the beginning, the priority which one to minimize was given to *Penalty1* using weight coefficients in fitness function.

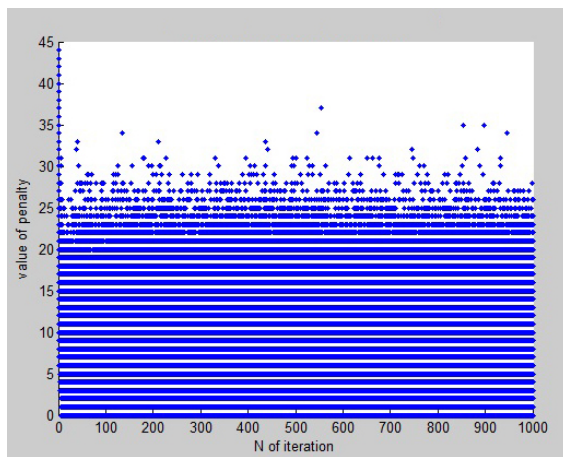


Figure 6.2: Number of planes scheduled to be serviced out of workers' shift

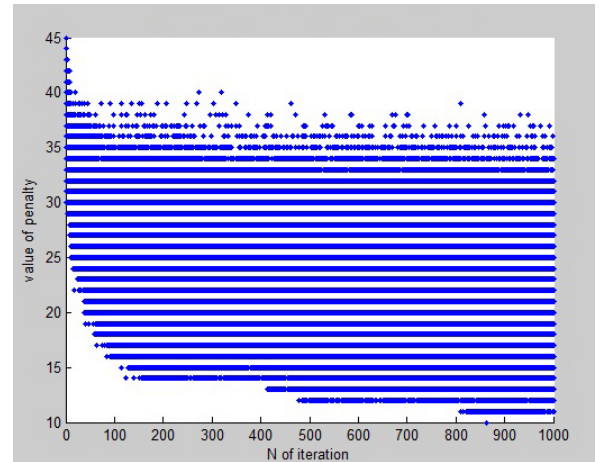


Figure 6.3: Number of planes scheduled to be serviced out of required time windows

6.2.2 Computational Time Analysis

The execution time of the algorithm run for workers of qualification "refueler" with the parameters described above is 11,4 minutes.

In order to measure on which functions the code spends the most time Profiler - graphical user interface provided by Matlab - was used. Results, from which it is easy to see the most time-consuming functions are shown on Figure 6.4.






Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
GeneticOperators	999	441.212 s	431.917 s	
Calculate_start_end	1000	334.095 s	334.095 s	
CalculateFitness	500000	226.728 s	13.038 s	
Penalty	500000	112.858 s	112.858 s	
WaitTravelTime	500000	100.832 s	100.832 s	
CreateMatingPool	999	51.736 s	31.104 s	
Tournament	499500	20.632 s	20.632 s	

Figure 6.4: Part of the Profiler report

Profiler helps to identify performance problems, such as:

- Unnecessary computation arised from oversight
- Costly functions, which could probably be replaced by others
- Recomputations, that could be avoided by storing results for future use

and is used to improve general performance of the algorithm.

Figures 6.5 and 6.6 present dependance of the run time of algorithm from number of iterations and the size of population respectively. Number of iterations that were taken: 50, 100, 250, 500, 800, 1000 and 1500, while population size was 500. Population size considered: 50, 100, 250, 500, 800 and 1000, while number of iterations was 1000.

Easy to notice that the dependance is linear. From Figure 6.4 is clear, that functions with longer execution time, such as *GeneticOperators* and *Calculate_start_end* are called once on iteration and function *CalculateFitness* which appeals to both *Penalty* and *WaitTravelTime* is called (the size of population * number of iterations) times. That explains the linear dependance.

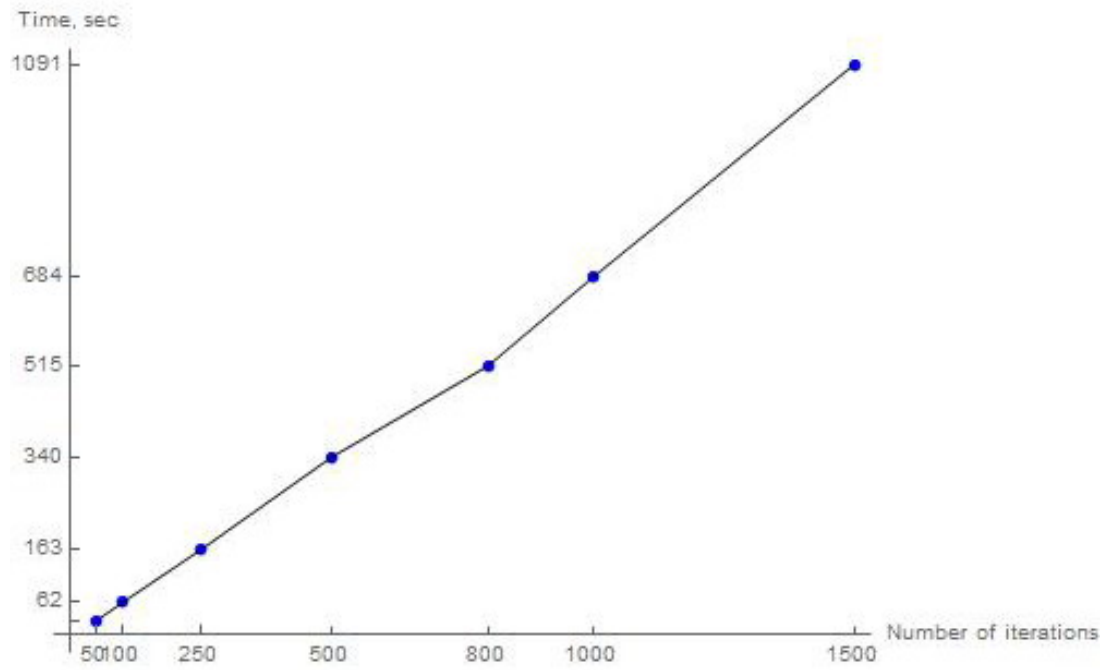


Figure 6.5: Diagram showing the relationship between run time of the algorithm measured in seconds and number of iterations

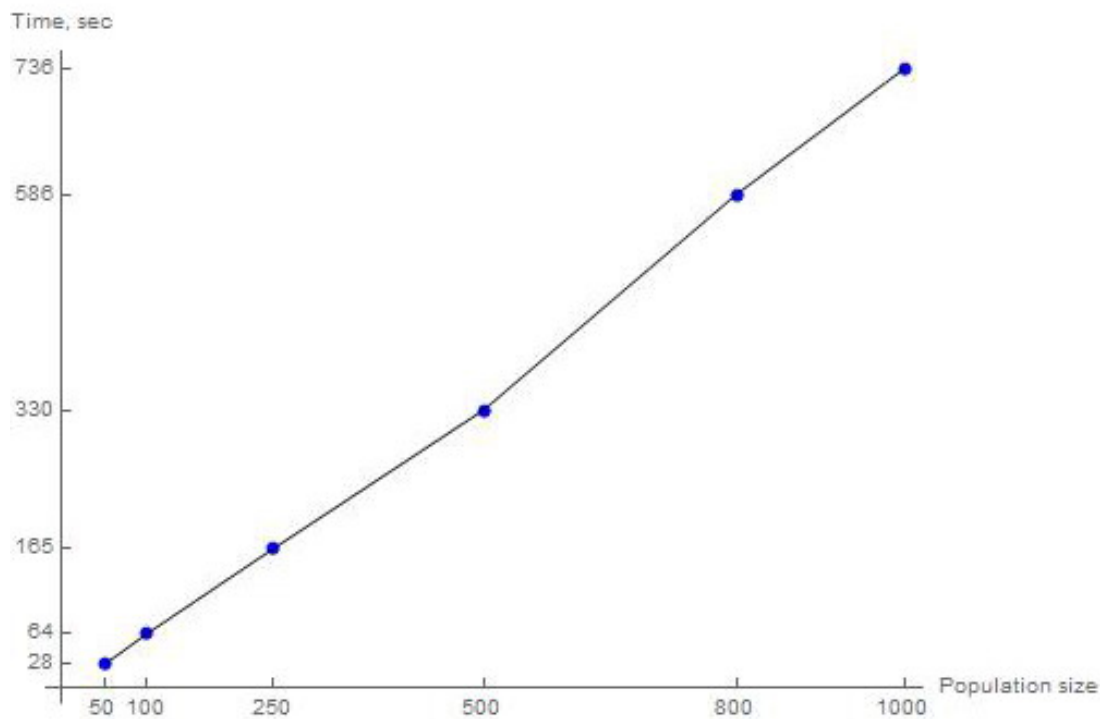


Figure 6.6: Diagram showing the relationship between run time of the algorithm measured in seconds and size of population

6.3 Case Study 2

6.3.1 Presentation of Model Output

In case study 2 values of the size of population and number of iterations could be taken much smaller comparant to the case study 1 since the algorithm is already working in a space of feasible solutions. The probability of mutation is quite high in order to introduce more diversity into population.

The algorithm was run with the following parameters:

$PSize = 100$	population size
$MaxIter = 200$	number of iterations
$\alpha_1 = 4$	weight for travel time in fitness function
$\alpha_2 = 2$	weight for waiting time in fitness function
$pMutation1 = 0.8$	probability of mutation1

for the same workers as in case 1 with the following working hours:

Name	Qualification	Tasks	Start	End
Juan	refueler	refueling	00600	01500
Alberto	refueler	refueling	00600	01500
Andreas	refueler	refueling	01200	02100
Christian	refueler	refueling	01200	02100
Olivier	refueler	refueling	01200	02100
Alexey	refueler	refueling	01200	02100
Vladimir	refueler	refueling	01900	10100
Rodrigo	refueler	refueling	01900	10100

The output of the model is performed as a schedule and is shown on Table 6.2.

Figure 6.7 shows tendency of fitness function of best individual in population on each iteration.

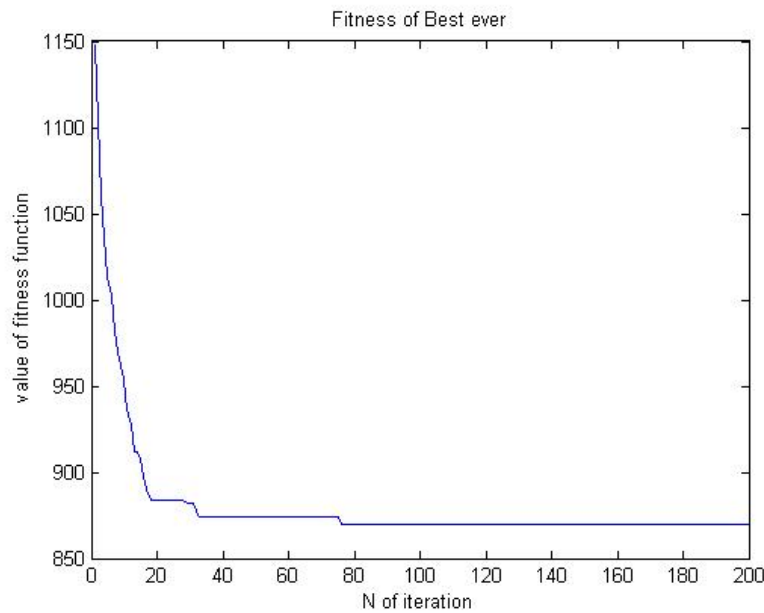


Figure 6.7: Fitness function of the best individual in population

Name	Qualification	Tasks	Start	End	Flight	Gate
Juan	refueler	refueling	0:06:48	0:07:01	200	2
Juan	refueler	refueling	0:07:12	0:07:30	1269	4
Juan	refueler	refueling	0:07:31	0:07:44	3165	3
Juan	refueler	refueling	0:07:52	0:08:10	6035	2
Juan	refueler	refueling	0:08:28	0:08:41	214	3
Juan	refueler	refueling	0:09:47	0:10:05	1044	3
Juan	refueler	refueling	0:10:12	0:10:30	5039	4
Juan	refueler	refueling	0:10:32	0:10:50	1036	2
Juan	refueler	refueling	0:11:12	0:11:30	100	3
Juan	refueler	refueling	0:11:48	0:12:01	105	2
Juan	refueler	refueling	0:12:08	0:12:21	7084	1
Alberto	refueler	refueling	0:06:23	0:06:36	3009	3
Alberto	refueler	refueling	0:06:47	0:07:05	5035	1
Alberto	refueler	refueling	0:08:23	0:08:36	1072	1
Alberto	refueler	refueling	0:09:03	0:09:16	190	2
Alberto	refueler	refueling	0:09:42	0:10:00	1034	1
Alberto	refueler	refueling	0:10:57	0:11:15	2098	1
Alberto	refueler	refueling	0:11:42	0:12:00	1038	4
Alberto	refueler	refueling	0:13:18	0:13:31	420	1
Andreas	refueler	refueling	0:13:48	0:14:01	5403	4
Andreas	refueler	refueling	0:15:12	0:15:30	689	1
Andreas	refueler	refueling	0:15:33	0:15:46	257	2
Andreas	refueler	refueling	0:16:12	0:16:30	3400	3
Andreas	refueler	refueling	0:17:53	0:18:06	1021	2
Andreas	refueler	refueling	0:18:58	0:19:11	698	2
Andreas	refueler	refueling	0:19:23	0:19:36	4067	3
Andreas	refueler	refueling	0:20:07	0:20:25	1356	2
Christian	refueler	refueling	0:12:48	0:13:01	354	2
Christian	refueler	refueling	0:13:48	0:14:01	288	3
Christian	refueler	refueling	0:15:08	0:15:21	5544	4
Christian	refueler	refueling	0:16:42	0:17:00	345	2
Christian	refueler	refueling	0:17:47	0:18:05	540	1
Christian	refueler	refueling	0:18:08	0:18:21	1277	3
Christian	refueler	refueling	0:18:42	0:19:00	7000	4
Christian	refueler	refueling	0:20:03	0:20:16	9226	4
Olivier	refueler	refueling	0:12:37	0:12:55	1266	3
Olivier	refueler	refueling	0:13:57	0:14:15	657	2
Olivier	refueler	refueling	0:15:08	0:15:21	4000	3
Olivier	refueler	refueling	0:16:12	0:16:30	266	4
Olivier	refueler	refueling	0:16:33	0:16:51	1081	1
Olivier	refueler	refueling	0:18:58	0:19:11	1002	1
Olivier	refueler	refueling	0:20:08	0:20:21	990	1
Olivier	refueler	refueling	0:20:33	0:20:46	134	3
Vladimir	refueler	refueling	0:21:13	0:21:26	2587	1
Vladimir	refueler	refueling	0:21:27	0:21:40	2087	2
Vladimir	refueler	refueling	0:22:18	0:22:31	1012	1
Vladimir	refueler	refueling	0:22:42	0:23:00	3210	2
Rodrigo	refueler	refueling	0:21:38	0:21:51	3008	4
Rodrigo	refueler	refueling	0:23:23	0:23:36	344	1

Table 6.2: Output of the algorithm

6.3.2 Computational Time Analysis

The execution time of the algorithm run for workers of qualification "refueler" with the parameters described above is 9 minutes.

The most time-consuming functions are shown on Figure 6.8 - screenshot from Profiler report. As in case 1 the longest takes the function *GeneticOperators*, though most of the time is spent by the functions it appeals to. The function *CheckTAS* verifies that the time windows and workers' shift constraints are satisfied. It is the most important feature of the implementation of the algorithm in case study 2, that is the reason why it is called such a big number of times. The function *generateInitialPopulation* takes significant time, since the algorithm builds the first feasible solution.









<u>Function Name</u>	<u>Calls</u>	<u>Total Time</u>	<u>Self Time</u> *	Total Time Plot (dark band = self time)
GeneticOperators	199	516.446 s	66.529 s	
CheckTAS	1514481	453.149 s	138.605 s	
Calculate_start_end_2	1514481	309.087 s	309.087 s	
Calculate_start_end	199	13.888 s	13.888 s	
ismembc (MEX-file)	1514071	5.457 s	5.457 s	
generateInitialPopulation	1	4.746 s	1.514 s	
CalculateFitness	20000	4.478 s	0.344 s	
WaitTravelTime	20000	4.134 s	4.134 s	

Figure 6.8: Part of the Profiler report

Figures 6.9 and 6.10 present dependance of the run time of algorithm from number of iterations and the size of population respectively. Number of iterations that were taken: 50, 75, 100, 150, 200 and 250, while population size was 100. Population size considered: 50, 75, 100, 150, 200 and 250, while number of iterations was 200.

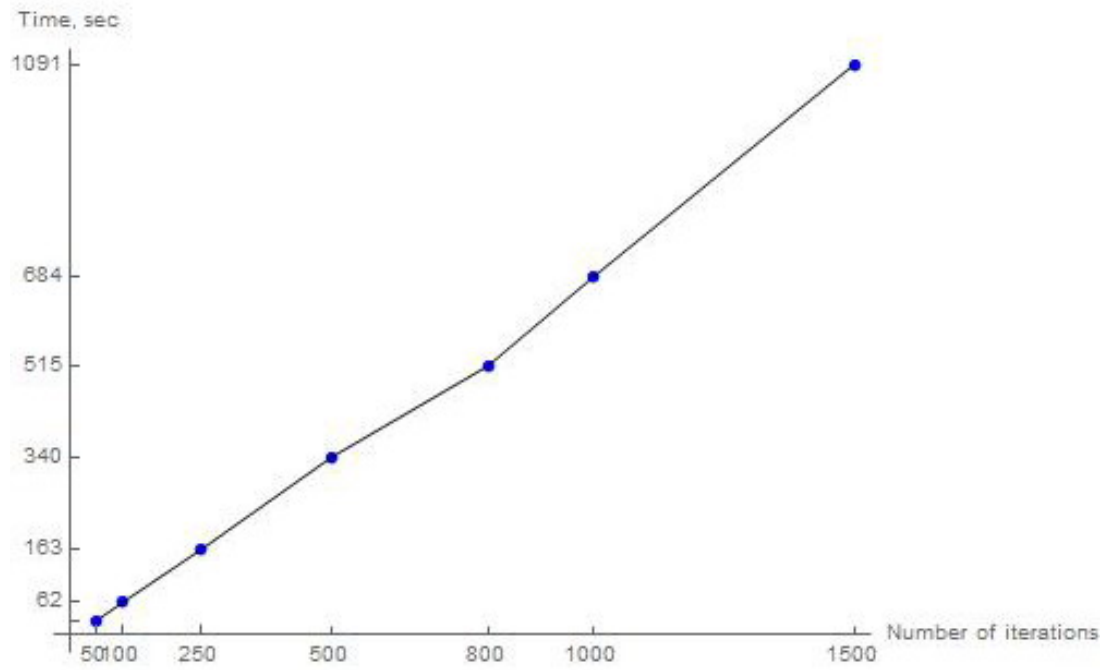


Figure 6.9: Diagram showing the relationship between run time of algorithm measured in seconds and number of iterations

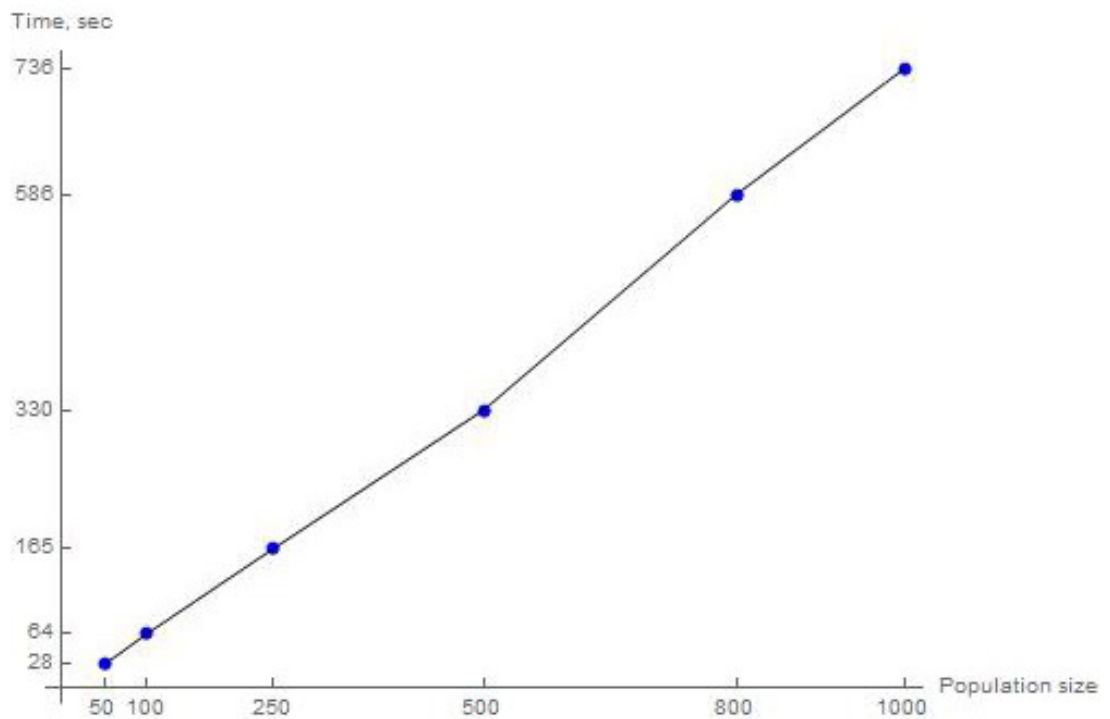


Figure 6.10: Diagram showing the relationship between run time of algorithm measured in seconds and size of population

Chapter 7

Analysis of Results

In case study 1 the algorithm did not give satisfactory results, it was not able to find even the feasible solution - schedule such that all aircraft are serviced according to specified by Service Level Agreement time windows.

The main reason for such a performance is that Evolution Strategies is a random search, therefore it is in need of a large search space and big number of iterations. The drawback of ensuring that is clearly the increased computational time which will make the algorithm no longer beneficial for operational and real-time planning.

In this thesis, in order to improve the performance of algorithm in case study 1, the attempt was made to use an insertion heuristic for initial population. However the experiment showed that it does not make efficiency any better, since the high probability of applied genetic operators renders it meaningless.

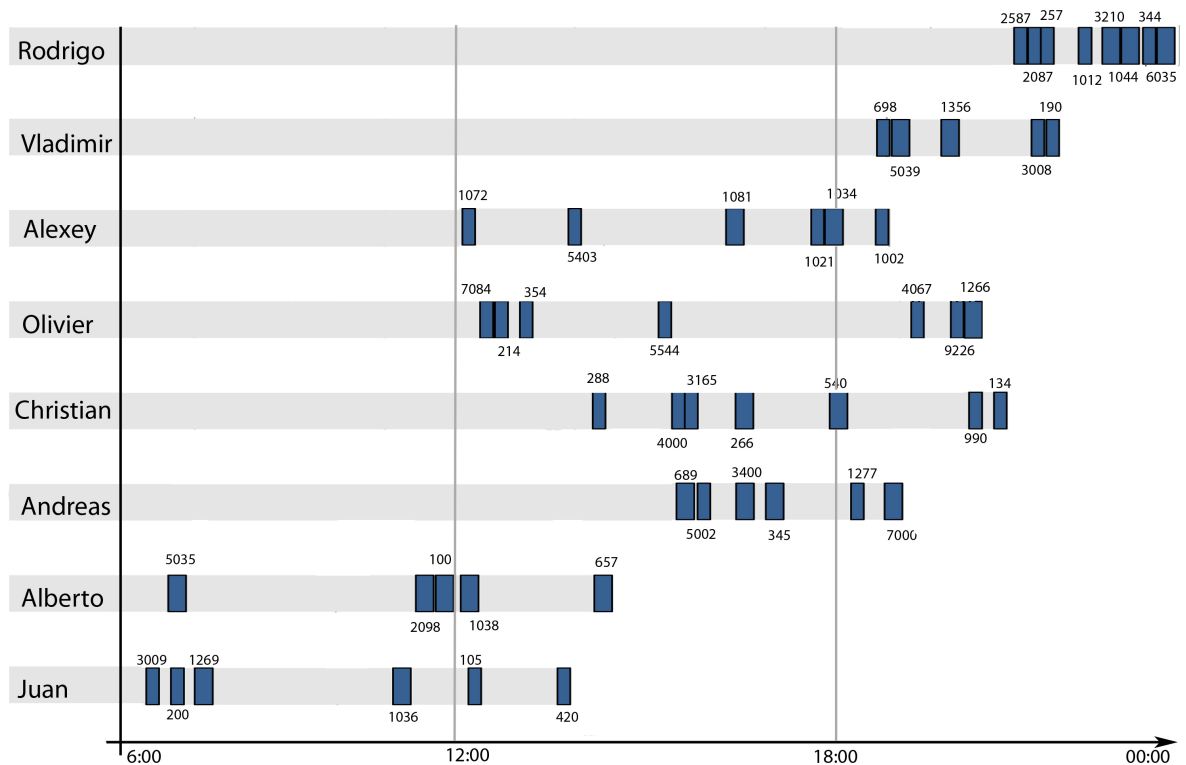


Figure 7.1: Gantt chart for the solution in case study 1

Figure 7.1 shows the solution obtained in case study 1 in the form of Gantt chart. It shows the sequence of tasks that should be performed by each worker distributed in time.

The implementation of Evolution Strategies considered in case study 2 coped well with the posed problem. It gave the feasible schedule for ground handling agents and it took comparable short time to finish (9 minutes). The Gantt chart for this solution is presented on Figure 7.2.

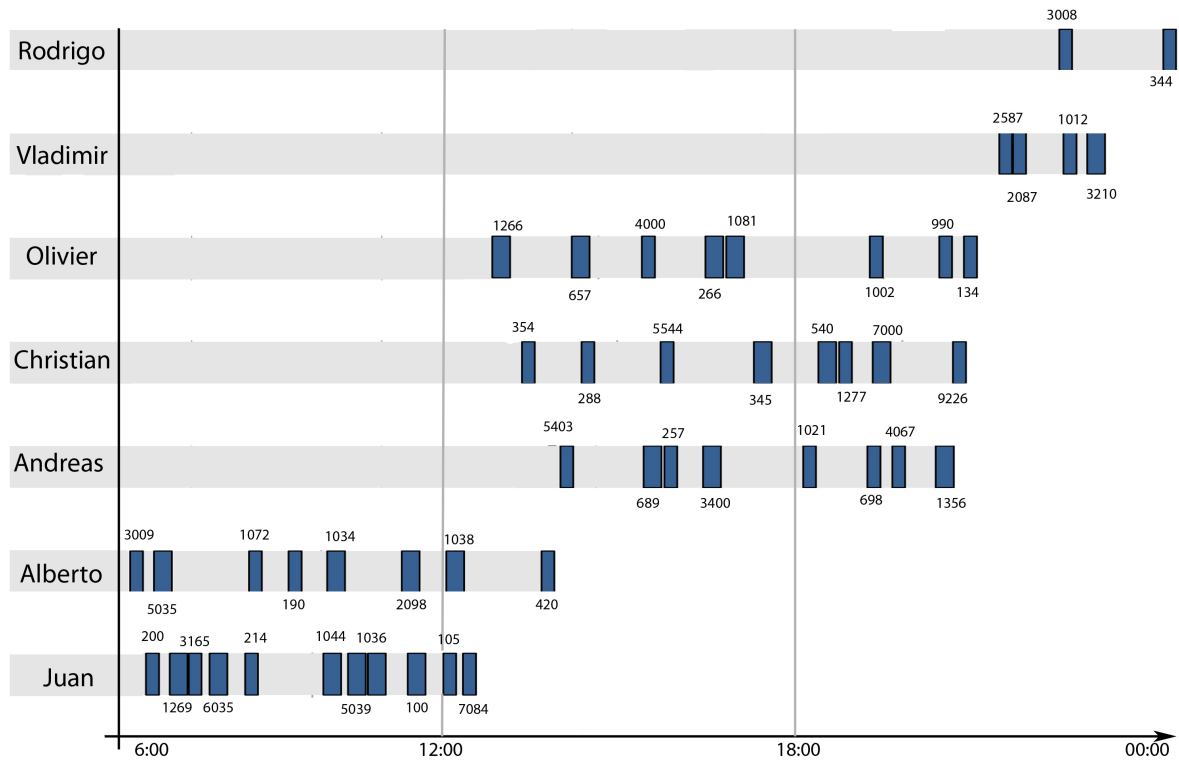


Figure 7.2: Gantt chart for the solution in case study 2

From Figure 7.1 and Figure 7.2 could be seen that in both cases shift constraints are satisfied for all the workers. However, in case study 2 tasks are distributed more uniformly, i. e. there are no longer big gaps in work of ground handlers. Moreover, the best solution found by the algorithm in case study 2 does not include schedule of the agent "Alexey", that means that given tasks could be performed by less number of workers than given initially providing that all necessary constraints are satisfied. Yet is another advantage in favor of the implementation of the algorithm in case study 2.

Chapter 8

Conclusions and Future work

8.1 Conclusions

All the goals stated at the beginning of this master thesis (Section 1.2) were achieved.

The first step of the work was devoted to analysis of different aspects of the ground handling staff scheduling problem: ground handling operations, their impact on flight delays, the economic aspects, etc. In this work, those aspects were explained in such a form that even people who are not familiar with the topic could get through it quickly, since it is crucial for correct understanding of the problem.

On the next step the mathematical model was formulated in the form of Vehicle Routing Problem with Time Windows (VRPTW). By definition the problem considered in this thesis closely fits the VRPTW problem statement, in which ground handling employees and aircrafts refer to vehicles and customers, respectively. The aircrafts have to be serviced at gates during time windows specified by Service Level Agreement and according to flight schedule. The algorithm was implemented in Matlab in a way that its main features – objective function and constraints – are easy to be changed.

After thorough studies of problem solving methods, the most appropriate one – Evolution Strategies – was chosen. There are several reasons for that decision. The main reason is that Evolutionary Algorithms operate on the big search space; they are working not only with one solution, but simultaneously with a population of solutions. Furthermore these algorithms are characterized by flexibility due to the possibility to introduce changes in formulation of the problem, mute on new parameters into the algorithm and add new constraints. The last but not least reason is the possibility of parallel computing with the usage of clusters.

The algorithm was built in Matlab and two case studies were implemented. The main difference between them is that the algorithm in the first case explores the whole solution space to build each solution of the next population (using two types of mutation operators), finally acting as a random search. In the second case it looks for each next solution only in the space of feasible solutions (using a single type of mutation). The run time of the algorithm in case 1 took 11,4 minutes to execute 1000 iterations, while the algorithm with the same input data in case 2 found better solution in 200 iterations taking 9 minutes. Comparative analysis in Chapter 7 explains advantages of the algorithm 2 implementation over the algorithm 1 in more details.

The approach proposed in this work could be an alternative option to the generic software packages widely used nowadays, such as the one presented by INFORM. The use of generic software has a drawback: the solution is often not fully adapted to the realities of particular ground handling company using it. On one side, scheduling system could include "extra" features; on another side, it could lack some. The algorithm considered in this master thesis is flexible and could be adapted to specifics of the company: changes in work rules, problem characteristics and sizes of operations. It should be noticed that run-time of the algorithm can be significantly decreased when passing the code to C++ or Java (Matlab is often used for prototyping).

8.2 Future Work

There are some concepts that could be implemented in the future in order to improve the results of the algorithm described in this master thesis. One of the ideas is to use hybridization of Evolution Strategies with some other algorithms presented in Chapter 4. The other idea will be the usage of local heuristics, f.e. in creation of initial population or in genetic operations.

With an eye to reduce the runtime, the same algorithm could be implemented in another programming language, such as C++ or Java. Since Evolution Strategies are using populations of independent individuals, parallel computing on different clusters could be helpful to make computations for each individuals, which could help to save some of the computational time.

Full realistic data from different ground handling companies could be collected in order to fully analyse the performance of the proposed algorithm and measure its utility in operational efficiency of ground handlers.

Bibliography

- [1] Official Journal of the European Communities L 272, 25/10/1996, p. 36 – 45
- [2] Slate Magazine: *"The Southwest Secret"*, by Seth Stevenson, June 12 2012
- [3] Eurocontrol: *"All-Causes Delay to Air Transport in Europe"*, April 2015
- [4] United States Department of Transportation, Bureau of Transportation Statistics
- [5] Boeing 777 airplane characteristics for airport planning
- [6] Yeung S. S. M., Yu I. T. S., Hui K. Y. L.: *"World at work: Aircraft cabin cleaning"*, Occupational and Environmental Medicine, 2005, p. 58 – 60
- [7] Kazda A., Caves R. E.: *"Airport Design and Operation"*, 2nd edition, Elsevier, Oxford, 2007, p. 183 – 184
- [8] Association of European Airlines: *"Recommendations for De-Icing/Anti-Icing of Aircraft on the Ground"*, 19th Edition, 2005
- [9] Marelli, S., Mattocks, G., Merry, R.: *"The role of computer simulation in reducing airplane turn time"*, AERO Magazine 1, 1998
- [10] Ashford N., Stanton H. P. M., Moore C. A.: *"Airport Operation"*, 2nd edition
- [11] Distribution Lab Analysis, Eurocontrol
- [12] Trabelsi S. F., Cosenza C. A. N., Cruz L. G. Z., Mora-Camino F., ICIEOM CIO: *"An operational approach for ground handling management at airports with imperfect information"*, 2013
- [13] Eurocontrol: *"Airport CDM Applications. Guide"*, July 2003, p. 15 – 17
- [14] SESAR factsheet N01/2011
- [15] Eurocontrol: *NM B2B Web Services. Web interface for system-to-system interoperability*, January 15
- [16] Nasser A. El-Sherbeny: *"Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods"*, Journal of King Saud University (Science), 2010, p. 123 – 131
- [17] Jayaraman Valadi, Patrick Siarry: *"Applications of Metaheuristics in Process Engineering"*, 2014
- [18] Bianchi L., Dorigo M., Gambardella L. M., Gutjahr W. J.: *"A survey on metaheuristics for stochastic combinatorial optimization"*, Natural Computing: an international journal 8 (2), 2009, p. 239 – 287.
- [19] Kirkpatrick S., Gelatt C. D., Vecchi M. P.: *"Optimization by simulated annealing"*, Science, May 1983, p. 671 – 680

- [20] Cerny V.: *"A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm"*, Journal of Optimization Theory and Applications, 1985, p. 41 – 51
- [21] Jason Brownlee: *"Clever Algorithms: Nature-Inspired Programming Recipes"*, 2014
- [22] Glover F.: *"Heuristics for Integer Programming Using Surrogate Constraints. Decision Sciences"*, 1977, p. 156 – 166
- [23] Glover F.: *"Future paths for integer programming and links to artificial intelligence"*, Comp. Oper. Res., 1986, p 533 – 549
- [24] Dreoj J., Petrowski A., Siarry P., Taillard E.: *"Metaheuristics for Hard Optimization: Methods and Case Studies"*, 2003
- [25] Baeck T., Schwefel H.-P.: *"An Overview of Evolutionary Algorithms for Parameter Optimization"*, Evolutionary Computation Volume 1, Number 1
- [26] Schwefel H.-P.: *"Numerische Optimierung von Computer-Modellen mittels der Evolutionssstrategie"*, 26. Birkhäuser, Basel, 1977

Annex A

```
profile -timer 'real'
profile on
clearvars P MatePool C P_best P_temp P_new Archive Fitness Fitness_C
Fitness_sort Fitness_sort_C CurrentWorkers CurrentTasks

%%% VALUES TO BE CHANGED ACCORDING TO INPUT
G = 4; % number of gates
W = 69; % number of workers
P = 50; % number of planes
A = 22; % number of SLA
Nexp = 1; % number of experiments
%CW = {'supervisor'; 'ramp_agent'; 'driver'; 'refueler'; 'servicer';
'cargo_handler'; 'baggage_handler'}; % list of workers qualifications
CW = {'refueler'};
depot_gate = G+1;

%%% Coefficients and probabilities
PSize = 100; % population size %PSizeA = [50, 75, 100, 150, 200, 250];
MaxIter = 200; %MaxIterA = [50, 75, 100, 150, 200, 250]; % number of
iterations
pTournament = 1;
TSize = 10; % tournament size
wTravel = 4; % weight for travel time in fitness function
wWait = 2; % weight for waiting time in fitness function
pMutation1 = 0.8; % mutation1 probability

%%% Reading input: types of gates, distances between them, workers structure,
flights structure, SLA
[type_gate, dist_gate, workers, flights, SLA] = getInput(G,W,P,A);

%%% Calculating initial time windows and saving it in the workload table as a
structure
[workload] = estimateWorkload(P,W,A,flights,workers,type_gate,SLA);
Ntasks = length(workload); % number of tasks

%for qqqqq=1:6 %PSize = PSizeA(qqqqq);
ss = RandStream('mcg16807','Seed',0);
```

```

    Archive(MaxIter)=struct('individual',[],'fitness',[],'waiting_time',[],
'travel_time',[]);
    [Fitness_sort_C, I] = sort(Fitness);
    P_best(1) = P(I(1));
    Archive(1) = P_best(1);

    %% iterations
    k=2;
    for Niter = 1:(MaxIter-1)
        [MatePool] = CreateMatingPool(PSize, P, pTournament, TSize);
        [C] = GeneticOperators(PSize, Nveh, pMutation1, depot_gate, dist_gate,
CurrentTasks, CurrentWorkers, MatePool);
        [C] = Calculate_start_end(C, PSize, Nveh, depot_gate, dist_gate,
CurrentTasks, CurrentWorkers);
        for i=1:PSize
            Individual = C(i).individual;
            [Waiting_Time(i), Travel_Time(i), Fitness(i)] =
CalculateFitness(Individual, Nveh, depot_gate, wWait, wTravel,dist_gate,
CurrentTasks, CurrentWorkers);
            Fitness_C(i) = Fitness(i);
            C(i).fitness = Fitness(i);
            C(i).waiting_time = Waiting_Time(i);
            C(i).travel_time = Travel_Time(i);
            Fitness(PSize+i) = P(i).fitness;
        end
        Wait_all = [Wait_all, Waiting_Time(1:PSize)];
        Travel_all = [Travel_all, Travel_Time(1:PSize)];

        %% updating P_best and Archive
        [Fitness_sort_C, I] = sort(Fitness_C);
        P_best(k) = C(I(1)); % P_best contains best individuals per population
        if Fitness_sort_C(1) < Archive(k-1).fitness
            Archive(k) = P_best(k);%Archive contains best individuals over
evolution
            k = k+1;
        else
            Archive(k) = Archive(k-1);
            k = k+1;
        end

        %% Merge P with C and take PSize of best, save them in P_new
        P_new(PSize) = struct('individual',[],'fitness',[],'waiting_time',[],
'travel_time',[]);

        %% Elitism mechanism: Selection of P based on Fitness of C and P
        [Fitness_sort, J] = sort(Fitness);

        %%If the algorithm finds a solution with penaltyShift equal to 0, the
population is restarted by duplicating this solution PSize times (but it does
it only once through all the iterations)
        if J(1) <= PSize
            P_temp = C(J(1));
        else
            P_temp = P(J(1)-PSize);
        end
        for i=1:PSize

```

```

        a = J(i);
        if a <= PSize
            P_new(i) = C(a);
        else
            a = a-PSize;
            P_new(i) = P(a);
        end
    end
    P = P_new;
%end
end

    Answer1(c) = Archive(MaxIter).fitness;
    Answer2(c) = Archive(MaxIter).waiting_time;
    Answer3(c) = Archive(MaxIter).travel_time;
end
end

    tElapsed = toc(tStart);
    disp(tElapsed);
    fprintf('          Fitness          WaitT          TravelT          \n');
    format short
    datasave = [Answer1 Answer2 Answer3];
    disp(datasave);
end

%%% Solution as timetable
Timetable(n) = Archive(Niter+1);
N = length(CurrentWorkers);
M = length(CurrentTasks);

for q=1:N
    L = length(Timetable(1,n).individual(1,q).route);
    for p=1:L
        T{s,1} = CurrentWorkers(1,q).name;
        T{s,2} = CurrentWorkers(1,q).qlf;
        T{s,4} = ConvertToTime(Timetable(1,n).individual(1,q).start_time(p));
        T{s,5} = ConvertToTime(Timetable(1,n).individual(1,q).end_time(p));
        for m = 1:M
            if Timetable(1,n).individual(1,q).route(p) == m
                T{s,3} = CurrentTasks(1,m).task;
                T{s,6} = CurrentTasks(1,m).flight;
                T{s,7} = CurrentTasks(1,m).gate;
                s = s+1;
            end
        end
    end
end
end
s=1;
TT = dataset({T 'Name', 'Qualification', 'Task', 'Start', 'End', 'Flight',
'Gate'})
profile viewer
p = profile('info');

```

```

end
function [workload] = estimateWorkload(P,W,A,flights,workers,type_gate,SLA)

j=1;
workload=struct('task',{},'qlf',{},'number_workers',{},'flight',{},'duration',
{'},'op_start',{},'op_end',{},'gate',{});
for k = 1:P
    current_STA = flights(k).STA;
    current_STD = flights(k).STD;
    current_type_gate = type_gate(flights(k).gate_number);
for i = 1:A
    if (strcmp(flights(k).airline,SLA(i).airline) == 1) &&
(strcmp(flights(k).type_plane,SLA(i).type_plane) == 1)
        % if gate is finger, not reading bus drivers
        if (current_type_gate == 0) && (strcmp(SLA(i).task,'bus') == 0)
            workload(j).task = SLA(i).task;
            for n=1:W
                if (strcmp(workload(j).task,workers(n).task{1}) == 1) ||
(strcmp(workload(j).task,workers(n).task{2}) == 1)
                    workload(j).qlf = workers(n).qlf;
                end
            end
            workload(j).number_workers = SLA(i).number_workers;
            workload(j).flight = flights(k).flight_arr;
            workload(j).duration = SLA(i).duration;
            workload(j).gate = flights(k).gate_number;
            if (strcmp('STA',SLA(i).initial_start_trigger) == 1)
                workload(j).op_start = current_STA + SLA(i).initial_start_time;
            elseif (strcmp('STD',SLA(i).initial_start_trigger) == 1)
                workload(j).op_start = current_STD + SLA(i).initial_start_time;
            end
            if (strcmp('STD',SLA(i).initial_end_trigger) == 1)
                workload(j).op_end = current_STD + SLA(i).initial_end_time;
            elseif (strcmp('STA',SLA(i).initial_end_trigger) == 1)
                workload(j).op_end = current_STA + SLA(i).initial_end_time;
            end
            j=j+1;
        % if gate is remote
        elseif (current_type_gate == 1)
            workload(j).task = SLA(i).task;
            for n=1:W
                if (strcmp(workload(j).task,workers(n).task{1}) == 1) ||
(strcmp(workload(j).task,workers(n).task{2}) == 1)
                    workload(j).qlf = workers(n).qlf;
                end
            end
            workload(j).number_workers = SLA(i).number_workers;
            workload(j).flight = flights(k).flight_arr;
            workload(j).duration = SLA(i).duration;
            workload(j).gate = flights(k).gate_number;

        % calculating initial time windows
        if (strcmp('STA',SLA(i).initial_start_trigger) == 1)
            workload(j).op_start = current_STA + SLA(i).initial_start_time;
        elseif (strcmp('STD',SLA(i).initial_start_trigger) == 1)
            workload(j).op_start = current_STD + SLA(i).initial_start_time;
        end
    end
end
end

```

```

    % calculating initial time windows
    if (strcmp('STD',SLA(i).initial_end_trigger) == 1)
        workload(j).op_end = current_STD + SLA(i).initial_end_time;
    elseif (strcmp('STA',SLA(i).initial_end_trigger) == 1)
        workload(j).op_end = current_STA + SLA(i).initial_end_time;
    end
    j=j+1;

end
end
end
end
end

function [P] = generateInitialPopulation(PSize, Nveh, depot_gate, dist_gate,
CurrentTasks, CurrentWorkers)

P = struct([]);
Possible.route = struct([]);
Possible.pos_task = struct([]);
Possible.task = struct([]);
for i = 1:PSize
    P(i).individual = struct([]);
    for j=1:Nveh
        P(i).individual(j).route = [];
        P(i).individual(j).task = [];
    end
    Ntasks = length(CurrentTasks);
    Todo = [1:Ntasks];
    length_Todo = length(Todo);
    Stock = [];
    for j=1:Nveh
        % save possible tasks
        Possible.worker = j;
        n = length(CurrentWorkers(j).task);
        Possible.pos_task = double(CurrentWorkers(j).task{1}(1));
        if n>1
            for v = 2:n
                if isempty(CurrentWorkers(j).task{v})==0
                    Possible.pos_task = [Possible.pos_task;
double(CurrentWorkers(j).task{v}(1))];
                end
            end
        end
        P(i).individual(j).pos_task = Possible.pos_task;
    for k=1:length_Todo
        Possible.route = [P(i).individual(j).route; Todo(k)];
        % add tasks
        m = length(Possible.route);
        if m>0
            ind = Possible.route(1);
            Possible.task = double(CurrentTasks(ind).task(1));
            if m>1
                for u = 2:m
                    ind = Possible.route(u);

```

```

        Possible.task = [Possible.task;
double(CurrentTasks(ind).task(1))];
    end
end
end
% check constraints
[ans1, Possible] = CheckTAS(Possible, depot_gate, dist_gate,
CurrentTasks, CurrentWorkers);
    if ans1 == 0
        P(i).individual(j).route = Possible.route;
        P(i).individual(j).start_time = Possible.start_time;
        P(i).individual(j).end_time = Possible.end_time;
        P(i).individual(j).task = Possible.task;
    else
        Stock = [Stock; Todo(k)];
    end
end
    Todo = Stock;
    length_Todo = length(Todo);
    Stock = [];
    Possible.route = struct([]);
    Possible.pos_task = struct([]);
end
    if isempty(Todo) ~= 0
        msg = 'Error: too many tasks.';
        error(msg)
    end
end
end
end

function [ans1, Current] = CheckTAS(Current, depot_gate, dist_gate,
CurrentTasks, CurrentWorkers)

ans1 = 0;
m = length(Current.route);
i = Current.worker;
[Current] = Calculate_start_end_2(Current, depot_gate, dist_gate,
CurrentTasks, CurrentWorkers);
    if m>0
        current_dist = dist_gate(CurrentTasks(Current.route(m)).gate,
depot_gate); % time from gate to depot
        current_late_end = CurrentWorkers(i).shiftend - current_dist; % time
worker should leave gate for depot
        for j = 1:m
            if Current.end_time(j) > CurrentTasks(Current.route(j)).op_end
                ans1 = ans1 + 1; % finishing after TW penalty
            end
            if Current.end_time(j) > current_late_end
                ans1 = ans1 + 1; % finishing after shift ends penalty
            end
        end
        h = ismembc(Current.task, sort(Current.pos_task));
        if ~all(h) == 1
            ans1 = ans1 + 1; % not being able to do this task penalty
        end
    end
end
end

```

```
function [P] = Calculate_start_end(P, PSize, Nveh, depot_gate, dist_gate,
CurrentTasks, CurrentWorkers)
```

```

for k = 1:PSize % going through whole population
for i = 1:Nveh % going through all vehicles
    Current_route = P(k).individual(i).route;
    P(k).individual(i).start_time = [];
    P(k).individual(i).end_time = [];
    m = length(Current_route);
    if m>0
        for j = 1:m
            if j>1
                current_end_time_prev = End_time(j-1);
                current_dist = dist_gate(CurrentTasks(Current_route(j-
1)).gate,CurrentTasks(Current_route(j)).gate);
            else
                current_end_time_prev = CurrentWorkers(i).shiftstart;
                current_dist =
dist_gate(depot_gate,CurrentTasks(Current_route(j)).gate);
            end
            current_duration = CurrentTasks(Current_route(j)).duration;
            current_early_start = current_end_time_prev + current_dist;
            current_op_start = CurrentTasks(Current_route(j)).op_start;
            Start_time = max(current_early_start, current_op_start);
            P(k).individual(i).start_time(j) = Start_time;
            current_op_end = CurrentTasks(Current_route(j)).op_end;
            End_time(j) = Start_time + current_duration;
            P(k).individual(i).end_time(j) = End_time(j);
        end
    end
end
end
end

```

```
function [Current] = Calculate_start_end_2(Current, depot_gate, dist_gate,
CurrentTasks, CurrentWorkers)
```

```

    Current.start_time = [];
    Current.end_time = [];
    m = length(Current.route);
    i = Current.worker;
    if m>0
        for j = 1:m
            if j>1
                current_end_time_prev = End_time(j-1);
                current_dist = dist_gate(CurrentTasks(Current.route(j-
1)).gate,CurrentTasks(Current.route(j)).gate);
            else
                current_end_time_prev = CurrentWorkers(i).shiftstart;
                current_dist =
dist_gate(depot_gate,CurrentTasks(Current.route(j)).gate);
            end
            current_duration = CurrentTasks(Current.route(j)).duration;
            current_early_start = current_end_time_prev + current_dist;
            current_op_start = CurrentTasks(Current.route(j)).op_start;

```

```

        Start_time = max(current_early_start, current_op_start);
        Current.start_time(j) = Start_time;
        current_op_end = CurrentTasks(Current.route(j)).op_end;
        End_time(j) = Start_time + current_duration;
        Current.end_time(j) = End_time(j);
    end
end

end

function [MatePool] = CreateMatingPool(PSize, P, pTournament, TSize)

MatePool(PSize) = struct('individual',[],'fitness',[],'waiting_time',[],
    'travel_time',[]);
rands = zeros(TSize);
for i = 1:PSize
    for j = 1:TSize
        rands(j) = randi(PSize);
    end;
    [Winner] = Tournament(P, rands, pTournament);
    MatePool(i) = Winner;
end
end

function [Winner] = Tournament(P, rands, pTournament)

R = length(rands);
Fitness = zeros(1,R);
for i=1:R
    Fitness(i) = P(rands(i)).fitness;
end
rand3 = rand;
[~, J] = sort(Fitness);
if rand3<pTournament
    Winner = P(J(1));
else
    Winner = P(J(2));
end
end
end

```

```

function [C] = GeneticOperators(PSize, Nveh, pMutation1, depot_gate,
dist_gate, CurrentTasks, CurrentWorkers, MatePool)
%Mutation = swapping of customers between routes or inserting customer to
%route of other vehicle

C = MatePool;

for i = 1:PSize
    % swapping of customers between routes
    for j=1:Nveh
        randM = rand;
        if randM <= pMutation1
            Current = C(i).individual(j);
            Current.worker = j;
            m = length(Current.route);

            if m>0
                rand1 = randi(m);
                rand2 = randi(Nveh);
                while rand2 == j;
                    rand2 = randi(Nveh);
                end
                a = Current.route(rand1);
                aa = Current.task(rand1);
                Current2 = C(i).individual(rand2);
                Current2.worker = rand2;
                n = length(Current2.route);

                iter = randperm(n,n); % full random string
                for uu=1:n
                    u = iter(uu);
                    b = Current2.route(u);
                    bb = Current2.task(u);
                    Current2.route(u) = a;
                    Current2.task(u) = aa;
                    Current.route(rand1) = b;
                    Current.task(rand1) = bb;
                    [ans1, Current] = CheckTAS(Current, depot_gate,
dist_gate, CurrentTasks, CurrentWorkers);
                    [ans2, Current2] = CheckTAS(Current2, depot_gate,
dist_gate, CurrentTasks, CurrentWorkers);
                    if (ans1 == 0) && (ans2 == 0)
                        C(i).individual(j).route = Current.route;
                        C(i).individual(j).task = Current.task;
                        C(i).individual(rand2).route = Current2.route;
                        C(i).individual(rand2).task = Current2.task;
                        break;
                    else
                        Current.route = C(i).individual(j).route;
                        Current.task = C(i).individual(j).task;
                        Current2.route = C(i).individual(rand2).route;
                        Current2.task = C(i).individual(rand2).task;
                    end
                end

                % inserting customer to another route

```

```

Current.route = Current.route([1:(rand1-1),
(rand1+1):m]);
Current.task = Current.task([1:(rand1-1),
(rand1+1):m]);
m = length(Current.route);
[~, Current] = CheckTAS(Current, depot_gate,
dist_gate, CurrentTasks, CurrentWorkers);

% adding at the beginning
Current2.route = [a, Current2.route]';
Current2.task = [aa, Current2.task]';
[ans3, Current2] = CheckTAS(Current2, depot_gate,
dist_gate, CurrentTasks, CurrentWorkers);
if ans3 == 0
    C(i).individual(j).route = Current.route;
    C(i).individual(j).task = Current.task;
    C(i).individual(rand2).route = Current2.route;
    C(i).individual(rand2).task = Current2.task;
    break;
else
    Current2.route =
C(i).individual(rand2).route;
    Current2.task = C(i).individual(rand2).task;
end

% adding in between
if n>1
    iter2 = randperm(n-1,n-1); % full random string
    for vv=1:(n-1)
        v = iter2(vv);
        Current2.route = [Current2.route(1:v)', a,
Current2.route((v+1):n)']';
        Current2.task = [Current2.task(1:v)', aa,
Current2.task((v+1):n)']';
        [ans4, Current2] = CheckTAS(Current2, depot_gate,
dist_gate, CurrentTasks, CurrentWorkers);
        if ans4 == 0
            C(i).individual(j).route = Current.route;
            C(i).individual(j).task = Current.task;
            C(i).individual(rand2).route = Current2.route;
            C(i).individual(rand2).task = Current2.task;
            break;
        else
            Current2.route =
C(i).individual(rand2).route;
            Current2.task = C(i).individual(rand2).task;
        end
    end
end

% adding in the end
Current2.route = [Current2.route', a]';
Current2.task = [Current2.task', aa]';
[ans5, Current2] = CheckTAS(Current2, depot_gate,
dist_gate, CurrentTasks, CurrentWorkers);
if ans5 == 0

```

```

        C(i).individual(j).route = Current.route;
        C(i).individual(j).task = Current.task;
        C(i).individual(rand2).route = Current2.route;
        C(i).individual(rand2).task = Current2.task;
        break;
    else
        Current2.route = C(i).individual(rand2).route;
        Current2.task = C(i).individual(rand2).task;
    end

    Current.route = C(i).individual(j).route;
    Current.task = C(i).individual(j).task;
    m = length(Current.route);

end
end
end
end
end
end

```

```

function [Waiting_Time, Travel_Time, Fitness] = CalculateFitness (Individual,
Nveh, depot_gate, wWait, wTravel, dist_gate, CurrentTasks, CurrentWorkers)

```

```

[Waiting_Time, Travel_Time] = WaitTravelTime(Individual, Nveh, depot_gate,
dist_gate, CurrentTasks, CurrentWorkers);
Fitness = wWait*Waiting_Time + wTravel*Travel_Time;
end

```

```

function [ans1, ans2] = WaitTravelTime(Individual, Nveh, depot_gate,
dist_gate, CurrentTasks, CurrentWorkers)

```

```

% ans1 = waiting time, ans2 = traveling time (max by vehicles, in numeric)

```

```

length_shift = zeros(1,Nveh);
sum_dur = zeros(1,Nveh);
Waiting_Time = zeros(1,Nveh);
Travel_Time = zeros(1,Nveh);
for i=1:Nveh
    Current_route = Individual(i).route;
    m = length(Current_route);
    % traveling time
    if m>0
        Travel_Time(i) = Travel_Time(i) +
dist_gate(depot_gate,CurrentTasks(Current_route(1)).gate); % time from depot
at the beginning
        Travel_Time(i) = Travel_Time(i) +
dist_gate(CurrentTasks(Current_route(m)).gate,depot_gate); % time to depot at
the end
        if m>1
            for j = 1:m-1
                Travel_Time(i) = Travel_Time(i) +
dist_gate(CurrentTasks(Current_route(j)).gate,CurrentTasks(Current_route(j+1))
.gate);
                % adding time between j and j+1
            end
        end
    end
end

```

```

end
% waiting time
length_shift(i) = CurrentWorkers(i).shiftend -
CurrentWorkers(i).shiftstart;
% length of shift
if m>0
    sum_dur(i) = CurrentTasks(Current_route(1)).duration;
    if m>1
        for j=2:m
            sum_dur(i) = sum_dur(i) +
CurrentTasks(Current_route(j)).duration;
        end
    end
    % sum of durations of tasks
    if sum_dur(i)+Travel_Time(i) < length_shift(i)
        Waiting_Time(i) = length_shift(i) - sum_dur(i) - Travel_Time(i);
    end
    %else
    %Waiting_Time(i) = length_shift(i);
end
end
ans1 = max (Waiting_Time);
ans2 = max (Travel_Time);
end
function [OutTA_Time, OutShift_Time, PenaltyTA, PenaltyShift] =
Penalty(Individual, Nveh, depot_gate, dist_gate, CurrentTasks, CurrentWorkers)

OutTA_Time = 0;
PenaltyTA = 0;
OutShift_Time = 0;
PenaltyShift = 0;
for i = 1:Nveh
    Current_route = Individual(i).route;
    m = length(Current_route);
    if m>0
        current_dist = dist_gate(CurrentTasks(Current_route(m)).gate,
depot_gate); % time from gate to depot
        current_late_end = CurrentWorkers(i).shiftend - current_dist; % time
worker should leave gate for depot
        for j = 1:m
            if Individual(i).end_time(j) >
CurrentTasks(Current_route(j)).op_end
                OutTA_Time = OutTA_Time + Individual(i).end_time(j) -
CurrentTasks(Current_route(j)).op_end;
                PenaltyTA = PenaltyTA + 1; % finishing after TW penalty
            end
            if Individual(i).end_time(j) > current_late_end
                OutShift_Time = OutShift_Time + Individual(i).end_time(j) -
current_late_end;
                PenaltyShift = PenaltyShift + 1; % finishing after shift ends
            end
        end
    end
end
end
end
end

```