universität wien

MAGISTERARBEIT

Titel der Magisterarbeit

Airport gate assignment with airline preferences: a robust approach for Vienna Intl. Airport

Verfasser

Gregor Gahbauer, Bakk.

angestrebter akademischer Grad

Magister der Sozial- und Wirtschaftswissenschaften
(Mag. rer. soc. oec)

Wien, 2012

Studienkennzahl lt. Studienblatt:	A 066 915
Studienrichtung lt. Studienblatt:	Magisterstudium Betriebswirtschaft
Betreuer:	o.Univ.Prof. Dipl.-Ing. Dr. Richard F. Hartl

ACKNOWLEDGEMENT

First, I would like to thank my advisor, o. Univ.–Prof. Dipl.–Ing. Dr. Richard F. Hartl, for his encouragement and his constructive feedback regarding my master thesis project.

I also want to thank Mag. Dipl.–Ing. Dr. Verena Schmid for her dedicated mentoring and support. Especially her ongoing motivation and creativity encouraged me a lot during this time.

Furthermore, I would like to thank my colleagues at the Chair of Production and Operations Management for their ideas and technical support.

I would also like to acknowledge the support of the following companies for providing me with the necessary data to accomplish this project:
Austro Control GmbH, Austrian Airlines AG, Niki Luftfahrt GmbH and Flughafen Wien AG.

In the end, I would like to thank my family and friends who supported me in various ways during my studies.

Vienna, June 2012

Contents

List of Figures

List of Tables

List of Abbreviations

ACG	Austro Control GmbH
AIP	Aeronautical Information Package
AOV	average objective value
AUA	Austrian Airlines
btw.	between
CEE	Central and Eastern Europe
CPLEX	IBM ILOG CPLEX optimizer (version 12.1)
CVRP	capacitated vehicle routing problem
EBOT	estimated block–on time
e.g.	for example
EOBT	estimated off–block time
EU	European Union
IATA	International Air Transport Association
ICAO	International Civil Aviation Organization
LNS	Large Neighborhood Search
MIP	mixed integer program
NLY	Niki Luftfahrt GmbH
QAP	quadratic assignment problem
RAGA–AP	robust approach for the airport gate assignment problem with airline preferences
RMS	resource management system
STDEV	standard deviation
VAH	Vienna Aircraft Handling GmbH
VIE	Vienna Intl. Airport
VRPTW	vehicle routing problem with time windows

1 Introduction

Since the beginning of the 20^{th} century, civil aviation has experienced an overall continuous growth in demand. The peak was reached in the 1950s/1960s with annual growth rates of up to 15%. Even during the 1990s, the industry experienced average rates of 6%.
Due to the fact that airports interact with passengers (representing demand) and airlines (representing supply), they are seen as the bottlenecks of the aviation business since capacity shortages can be identified at a very early stage. Airport managers very often have to deal with existing infrastructure which cannot be changed or adapted easily. Therefore, the field of research of *airport logistics* has gained in importance during the last decades [17].

One particular research topic is the *airport gate assignment problem* which optimizes the assignment of flights (demand) to airport gates (supply). The problem itself is considered to be hard to solve since many involved stakeholders want to influence the decision making to their favor.

In general, airport gate assignment can be divided into seasonal, daily and tactical planning [8]. The model presented in this thesis belongs to daily planning, since aircraft turnarounds (linkage of in– and outbound flights) are already preassigned and no information about flight delays is yet available.

In the beginning, research was mainly focused on the minimization of passenger walking distances within an existing airport terminal [10–12]. During the last two decades, the focus was extended to other objectives which are more relevant in real life, such as robust scheduling with stochastic flight delays, disruption management, aircraft reassignment algorithms, etc. [17, 25, 26].

Problem description

The *airport gate assignment problem* developed in this thesis project – titled *robust approach for the airport gate assignment problem with airline preferences (RAGA–AP)* – can be formulated as a mixed integer program (MIP). In order to adapt this deterministic problem to real life situations including aircraft delays, a robust approach was chosen in order to reduce rescheduling during tactical planning. Robust scheduling of airport gates has become increasingly important during the last decades due to growing air traffic and subsequent tighter airline schedules.

In order to demonstrate the importance of robustness, let us consider the following example:

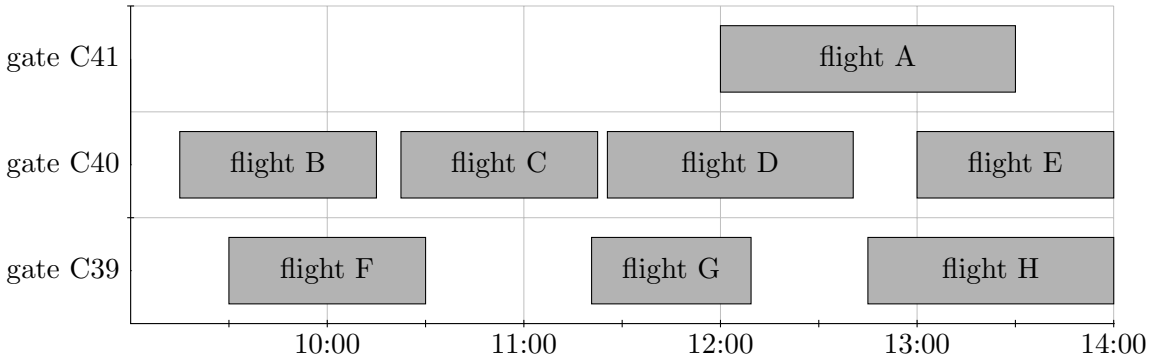


Figure 1: Necessity of robustness in gate assignment

Figure 1 shows an assignment of 8 flights to 3 gates. The small gap between flight C and D at gate C40 might lead to problems in case the first of these two flights is delayed or the second one arrives earlier than scheduled. A possibility to improve this situation would be to reassign flight C to gate C41. This reassignment would increase the robustness of the schedule.

Another important goal of real life gate assignment is the consideration of airline preferences. Besides an airline's desire to park at a certain gate area, the prefer-

ence value of assigning a flight to a specific gate is also influenced by other factors such as passenger convenience and handling costs:

passenger handling:

Network carriers attach great importance to passenger comfort. Therefore, such airlines prefer gates located at a terminal where passengers can board the aircraft by jetway. In case of Vienna Intl. Airport (VIE), this might interfere with fast boarding times, since jetway boardings use only one aircraft door, compared to apron gates[1] where two stairs are used.

Other reasons for the preference of gates with jetways are the existence of special security checks and customs offices in the terminal.

incurring costs for handling, parking and aircraft operations:

It is assumed that gates located on the apron have cheaper parking and handling fees than gates located at a pier. Lower costs might incur because of unnecessary pushbacks and less infrastructure at the gate (e.g. aircraft guidance via marshaller instead of docking–guidance system, passenger boarding via bus instead of jetways,...).

Another advantage of apron gates is the reduction of taxi time from the runway to the parking position and vice versa. At VIE, it is assumed that low–fare airlines like FlyNiki, AirBerlin and Germanwings prefer gates on the E–block, because those parking positions are located closer to the most used runways for departures (runways 29 and 16).

Due to these circumstances, airlines are often required to make trade–offs between passenger comfort and incurring costs.

In order to achieve a valid schedule, an airport gate assignment has to meet certain criteria, as for instance limitations in size, gate characteristics and limitations regarding neighboring gates:

[1] apron: area with remote parking positions serviced usually with buses instead of jetways

basic limitations: an aircraft can only be assigned to one gate. Furthermore, many flights can be assigned to the same gate, but not at the same time.

wingspan: the wingspan of an aircraft cannot exceed the gate width.

length: the length of an aircraft cannot exceed the gate length.

gate characteristics: at VIE, four different flight characteristics exist: Schengen, Non–Schengen, European Union (EU) and international flights. Each type has different regulations regarding customs and passport control. For simplification, these four flight characteristics were combined into three gate types in the model: pier gates (both Schengen and Non–Schengen) and apron gates. Flight and gate characteristics have to be equal in case of pier gates and can be unequal in case of apron gates, since it does not matter if the bus drives the passengers to the Schengen or Non–Schengen arrival area (considering travel time).

shadow restrictions: a popular method to increase the flexibility of available airport gates is the implementation of shadow restrictions which are used at all Austrian airports[2]. In case a shadow restriction exists between two gates, only one gate can be occupied at the same time, because aircraft wingtip clearance to the neighboring gate is not guaranteed [13].

Figure 2 shows an example of a shadow restriction at VIE. If an aircraft is parked at gate F05 (maximal aircraft wingspan: 65 meters), the wingtips reach far into the neighboring gate areas. Therefore, the neighboring gates have to be blocked until the aircraft leaves gate F05. On the other hand, if gates F03 or F09 (maximal aircraft wingspan: 36 meters each) are blocked, parking position F05 has to be blocked by a shadow restriction.

[2] see AIP Austria (Part III – AD 2 – Aircraft Parking/Docking Charts) for further information. Available online at http://eaip.austrocontrol.at.

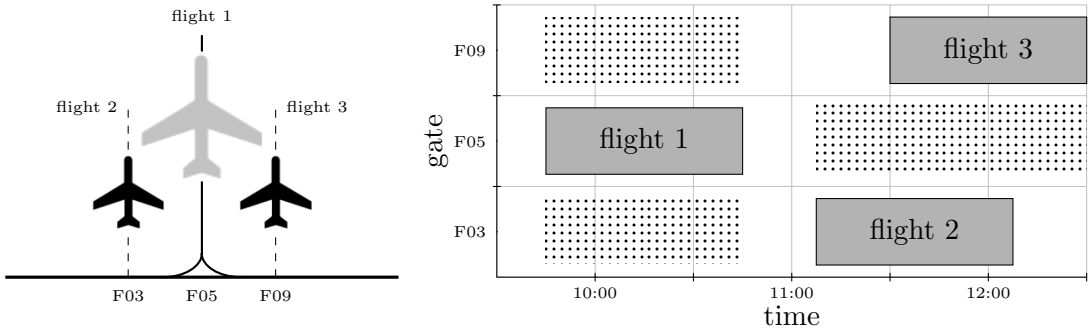


Figure 2: Example for gate shadow restrictions

Vienna International Airport – VIE

The airport was privatized in 1992 and is now owned by the province of Lower Austria (20%), the city of Vienna (20%), a private employee–participation foundation (10%) and private shareholders (50%).
In 2011, a total of 21.1 million passengers landed and/or departed to/from VIE, which is an increase of 7.2% compared to 2010. Thereof, 6.5 million (30.8%) are transfer passengers. VIE uses a two–runway system: runway 11/29 (3500 meters long, 45 meters wide) and runway 16/34 (3600 meters long, 45 meters wide). Although parallel approaches to both runways are only possible during good visibility and calm (southern) winds, the airport recorded 246,157 traffic movements in 2011. VIE has evolved into a major hub for destinations in CEE countries, thanks to the special focus of Austrian Airlines (AUA) [15].

The traffic at airports is often concentrated in so–called *waves*, which are time periods with concentrated in– and outbound traffic. Traffic waves often have an effect on the complexity of gate assignment, since different requirements (e.g. demand of gates) exist at different times.

Figure 3 shows the traffic progression at VIE on July 12$^{\text{th}}$ 2011 containing 7 traffic waves. This data indicates that a maximum of 53 aircraft are parked at

the airport at the same time. Since VIE features 115 gates, it is not a question of capacity, but of how to assign all aircraft in an efficient way.
A more detailed description of the structure of the 24–hour instance containing 401 turnarounds is given in Section 5.1.3.

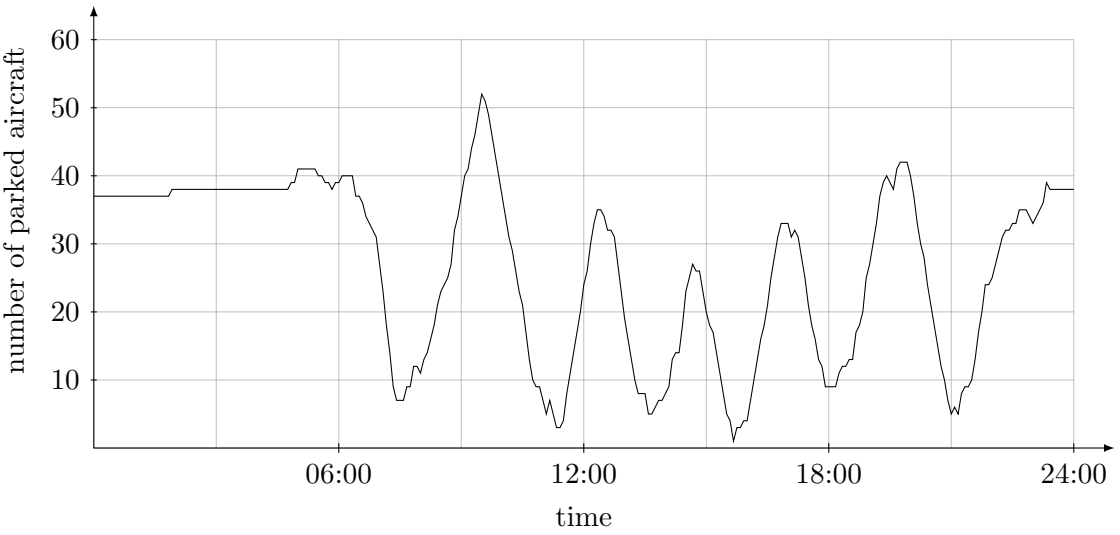


Figure 3: Vienna Airport traffic progression – July 12$^{\text{th}}$ 2011

VIE features two *general aviation centers* (east and west) where Vienna Aircraft Handling GmbH (VAH) handles all general aviation flights.
Parking positions at VIE which are close to each other are combined to *gate blocks*, which are numbered from A to K.
Parking positions located at *Pier West* (C–block) are used for flights to/from countries which signed the Schengen contract. *Pier East* (D–block) contains parking positions used mainly for Non–Schengen, Non–EU and international flights. B–, E– and F–blocks feature apron parking positions which are usually used by low–fare or non–*Star–Alliance* airlines[3]. The remaining gate blocks A, H and K are mainly used by cargo airlines and for parking aircraft. More information about

[3] F–parking positions are located at the new *SKYLINK* terminal which was not yet in operation during the master thesis project and therefore the positions are used as apron gates handled by buses.

the layout of VIE is depicted in the Aeronautical Information Package (AIP) Austria (LOWW AD 2.24–1–2) available online[4] or in the Appendix on page 57.

Software currently in use at Vienna Intl. Airport

The software used for gate assignment at VIE is part of the resource management system (RMS) *sally*, developed by the German company *Delair*[5].

Sally is also used at Zurich and Frankfurt/Main airport. In case of Zurich, the gate assignment of a 24–hour schedule was accelerated from more than 4 hours to approximately 15 seconds.

In case of flight delays and possible consequential conflicts, the RMS proposes possible flight gate changes automatically to the airport managers who can choose between different alternative assignments. Other advantages are the consideration of connecting passengers within a terminal (passenger walking distances) as well as the automatic linkage of in– and outbound flights which is very effective in case of sudden aircraft changes.

sally is divided into three different modules (gate planning, baggage planning and check–in planning) and includes a strategic (seasonal planning and what–if scenarios) as well as a tactical version (daily and tactical planning). The calculated assignment is depicted in a Gantt chart. Figure 4 shows the baggage module which assigns flights to baggage carousels at Zurich airport [7].

sally uses two types of rules [7]:

hard rules which have to be observed
(e.g. time–overlapping constraints, aircraft wingspan/length vs. gate size,...)

soft rules which improve the assignment if they are observed
(e.g. airline preferences, Schengen/Non-Schengen flights,...)

[4] http://eaip.austrocontrol.at
[5] http://www.delair.de

This set of rules can be individualized for specific local needs, depending on whether the airport is a regional or hub airport.

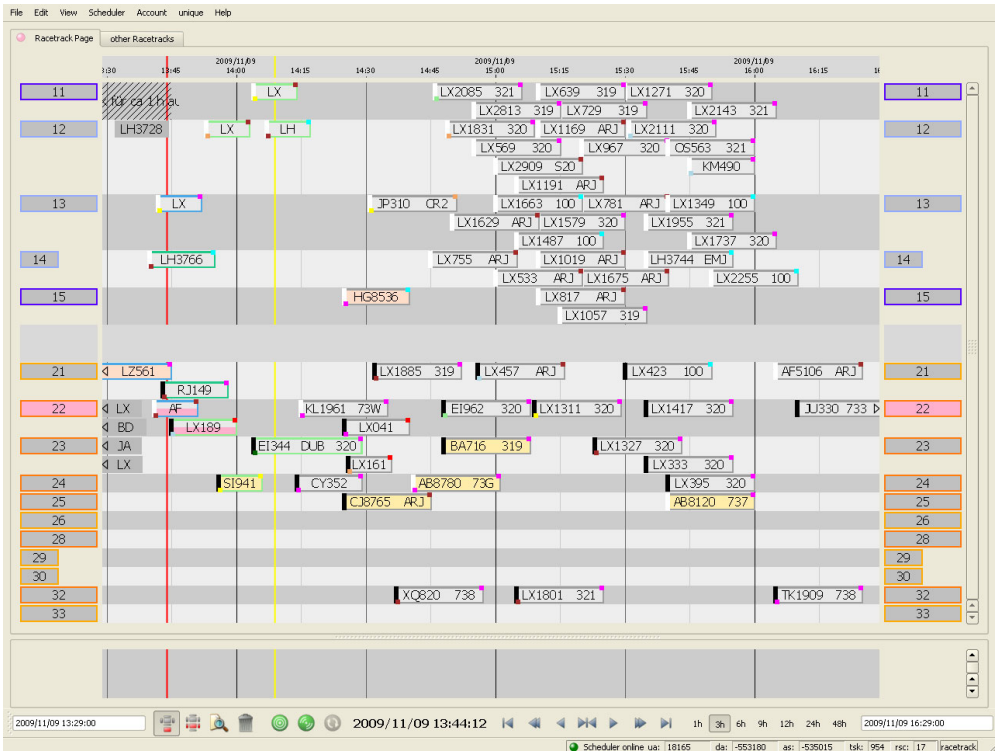


Figure 4: RMS sally – baggage module at Zurich airport[6]

According to *Delair*, the RMS is based on a constraint satisfaction problem solver which implements unary and binary constraints and uses constraint propagation, prioritization based on an assessment heuristic as well as backtracking to a conflict solution. Consequentially, every solution within the solution space is an equal and valid solution and no optimization of the problem is achieved (which explains the comparatively short computation time) [7].

[6]source: http://www.airport-int.com/upload/image_files/Sally-Racetrack.jpg last accessed: June 19th 2012

Finally, some terms and definitions used in this thesis are explained.
In the literature, the terms *gate*, *position* and *parking position* are often used in the same context. In practice, a *(parking) position* is the area where the aircraft is parked, whereas a *gate* is the area in a terminal where departing passengers are waiting for their flight. In this thesis, these terms are also used as synonyms.
A second case is the use of the terms *terminal* and *pier*. A *terminal* also includes the check–in area, passport control, security checks, duty–free shops etc. In contrast, a *pier* is located between the *terminal* and the apron and contains almost exclusively *gates*.

The research topic of this thesis is motivated by the scientific and practical relevance of the problem as well as personal interest after a four year employment within the department of *Aircraft Handling Services* at Vienna Intl. Airport.

This thesis is structured as follows:
Section 2 gives a short overview of past research developments in the area of airport gate assignment. The mathematical model is described in Section 3. Thereafter, Section 4 presents the solution method used to solve the MIP. At the very end, Section 5 presents the real life data used as input for this model as well as the computational results.

2 Literature review

The *airport gate assignment problem* in its original form can be formulated as a *quadratic assignment problem (QAP)*, in which the cost of assigning an aircraft to a specific gate depends on the passenger walking distance, the demand and the interaction with other gates [16].

In general, gate assignment problems can be divided into single and multiple slot models (see Figure 5). Single slot models – developed at the beginning of airport–oriented research – resemble the NP–hard QAP and consider a single time interval where at most one flight can be assigned to any gate. Multiple time slot problems were first introduced in [16] in 1998 and consider multiple time intervals to allow more than one aircraft at the same gate [12]. The model developed in this thesis belongs to the latter.

Dorndorf et al. [12] divide gate assignment research streams into two categories: rule–based expert systems and mathematical programming techniques (which will be discussed first).

Airline– and passenger–oriented objectives for static/deterministic problems (considering baggage carrying distances, walking distances or passenger delays) were deeply researched in the past; whereas airport–oriented objectives still offer many possibilities for future research.

According to [17], typical objectives are:

- minimize number of aircraft assigned to apron gates
- minimize total or average passenger walking distances
- minimize baggage transportation distances
- minimize total passenger delay
- minimize deviation of a current schedule to a reference schedule
- minimize number of towing procedures

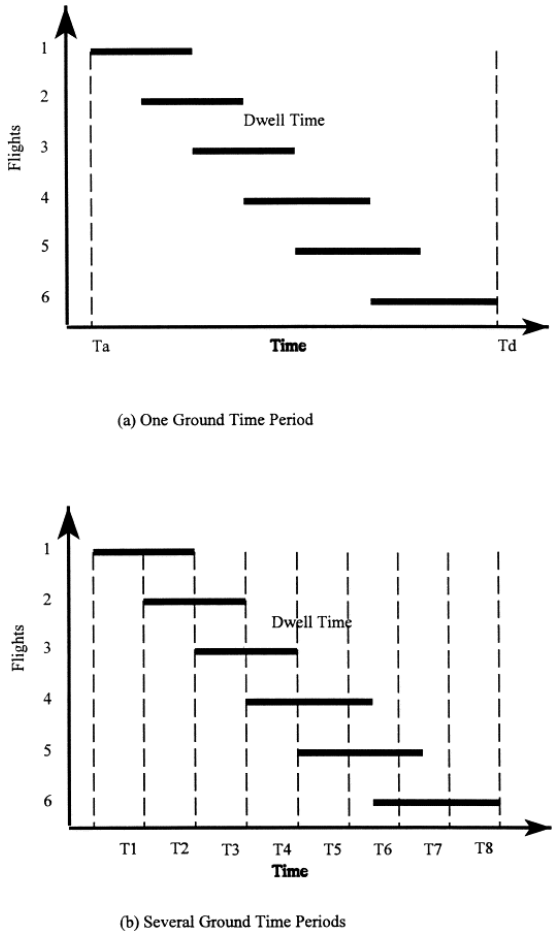


Figure 5: Single and multiple slot models (source: [16])

Between 1973 and 1977 Braaksma et al. [3–5] developed the first mathematical models which considered intra terminal travel of passengers. Babić et al. [1] presented a model for the single slot gate assignment problem minimizing walking distances between the gate and the entry/exit which was solved with an exact branch–and–bound algorithm. Later, [19] added transfer passengers to the model and solved the problem with LP–relaxation and greedy heuristics [12].

The approach including passenger walking distances is especially preferred by airlines within the same alliance, since short transfer distances are desired to reduce departure delays caused by walking passengers. A high number of flights located

at the terminal gives the passengers more comfort than gates serviced by buses. Airports, on the other hand, pursue different strategies. Although the minimization of walking distances might ensure the robustness of a schedule, airports are not always interested in guiding the passengers to their gate via the shortest path. A reason for this circumstance is the fact that airports do not only earn money through the handling of passengers, but also for renting advertising panels within a terminal to advertising agencies.

Chang [6] added a minimization of the baggage transfer distances at a hub airport and applied a simulated annealing algorithm to the model. A column generation algorithm was used in Diepen et al. [9] to solve the gate and bus assignment at Schipol Airport (Amsterdam, Netherlands).

A popular model based on [1] is the *over–constrained airport gate assignment problem* presented in Ding et al. [10, 11], in which the capacity of terminal gates is exceeded by the number of flights; hence apron gates were introduced. A closer explanation follows in Section 3.2.1.
Models regarding stochastic flight delays, robust scheduling and reassignment algorithms were developed in [2, 22, 24–26].
Jaehn [17] presents different models regarding disruption management, since previous research was mainly focused on static/deterministic problems and little attention was paid to this particular field of research.

The second research stream of airport gate assignment are simulations and rule–based expert systems. Compared to the category of mathematical programming techniques these systems seem to cope better with difficulties in case of multiple performance criteria and uncertain information.
Typical examples are systems to *"evaluate the effectiveness of operational options to improve the gate utilization"* [16], systems containing sensitivity analysis and systems for evaluating the effects of particular rules on gate utilization. To in-

crease the flexibility of a gate assignment, [12] proposes the implementation of mathematical programming techniques into simple expert systems.
Further information on the two research streams is presented in more detail in [12, 16].

The main focus of this thesis project is the mathematical model of the RAGA–AP solved with a MIP solver and a metaheuristic.

Dorndorf et al. [12] recommend an implementation of the Large Neighborhood Search (LNS) metaheuristic for real life problem sets with a large number of flights and gates as well as for manually generated small instances.
The RAGA–AP shows similarities to the *vehicle routing problem with time windows (VRPTW)* where the gates are denoted as routes and the aircraft turnarounds as customers with (hard) time windows.
LNS was originally presented in [23] in 1998 for the *capacitated vehicle routing problem (CVRP)* and the *VRPTW*. According to [21], LNS belongs to the class of *Very Large Scale Neighborhood Search (VLSN)*. This class is characterized by exponentially growing neighborhoods and neighborhoods which are too large to be searched in practice.
LNS has shown good results for the *VRPTW* in the past [23] and it seems as if it has never been applied to a variation of the *airport gate assignment problem*.

3 Mathematical model

In this section, the mathematical model developed in this thesis project will be introduced. After describing the notation used for the problem, an overview of the underlying model (*general (static) airport gate assignment problem*) is given. In the last section, the actual MIP for the *RAGA–AP* is presented.

3.1 Notation

We are going to define the following sets:
Let F denote the set of flights; more precisely the set of aircraft turnarounds. For every flight i, the subset F' contains a list of flights j where i precedes j, that is $p_{i,j}$=1 (see Table 2). *G(i)* is a subset of G which only includes suitable gates for flight i. In this context, suitable means that the limitations concerning wingspan, length and gate characteristics are met. The set A contains all relevant information regarding aircraft specifications. P represents the set of airline preferences which contains data concerning a flight's desire to be assigned to a specific gate. The set of shadow restrictions N contains pairs of gates where a shadow restriction exists. A summary of all the different sets used in the model is also given in Table 1.

Abbreviation	**Description**
F	set of flights
$F' \subseteq F$	subset of F with flights where flight i precedes flight j
G	set of gates
$G(i) \subseteq G$	subset of G with possible gates for flight i
A	set of aircraft
P	set of airline preferences
N	set of shadow restrictions

Table 1: Notation used for sets

Abbreviation	Description
$i,j \in F$	indices used to describe flights
$k,l \in G$	indices used to describe gates
n	number of flights
m	number of gates
$f_{i,k} \in [0,1]$	flight i possible on gate k
$s_{k,l} \in [0,1]$	shadow restriction between gate k and l
$p_{i,j} \in [0,1]$	flight i is preceding flight j
$u_{i,k} \in P$	preference value of assigning flight i to gate k
t_i^A	estimated block–on time
t_i^D	estimated off–block time
t_k^{setup}	setup time of gate k
t^{IV}	length of time interval in minutes
M	very high number
α_1, α_2, α_3	objective function weights

Table 2: Notation associated with (input–) parameters

Table 2 shows the notation used to describe the (input) parameters. $f_{i,k}$ equals 1 if the (hard) constraints concerning wingspan, length and gate characteristic are fulfilled. Airline preferences are represented in the model as $u_{i,k}$. The variable assumes a high negative value (e.g. –1000) in case flight i is not desired on gate k and a value between 0 and 100 in case of a desired assignment.

t_i^A and t_i^D represent the arrival and departure times of the flights. Since the model does not include taxi times to and from the runway, t_i^A equals the estimated block–on time (EBOT) and t_i^D the estimated off–block time (EOBT). This approach – a gate k is blocked from the time the aircraft touches down on the runway until it departs for the succeeding flight – is also used by VIE airport managers in real life.

t_k^{setup} represents the gate setup time. This variable includes the time needed to push the previous aircraft back from the parking position, the time to prepare the visual docking guidance system for the next flight, the time needed by the

ground handlers to move new equipment to the gate as well as the arrival of the succeeding aircraft [12]. In real life, every gate has different setup times depending on the location on the apron. In the presented model, this variable is equal for every gate.

t^{IV} represents the length of the time interval in which all included times are measured. This is especially necessary to standardize the times used in the flight set F.
e.g. if t^{IV}=5, one time unit equals 5 minutes and 01:00 equals 12 time units.
A closer explanation follows in Section 5.1.1.

Abbreviation	**Description**
$x_{i,k} \in [0,1]$	assignment of flight i to gate k
g_{min}	minimal gap (btw. any two consecutive flights at all gates)
t_k^{FA}	arrival time of first flight at gate k
t_k^{LD}	departure time of last flight at gate k
$x_{i,k}^{FF} \in [0,1]$	flight i is first flight at gate k
$x_{i,k}^{LF} \in [0,1]$	flight i is last flight at gate k

Table 3: Notation used for decision variables

Table 3 shows the decision variables used for the MIP.
$x_{i,k}$ denotes the assignment of flight i to gate k. The variable assumes the value 1 in case i is assigned to k and 0 otherwise. g_{min} represents the smallest gap between any two consecutive flights at all gates. The last four variables are used to calculate the total idletime between the flights. A closer explanation of the decision variables follows in Section 3.2.2.

3.2 Problem formulation

This section presents the mathematical formulation of the *airport gate assignment problem*. Section 3.2.1 explains briefly the classical problem which forms the basis of the model developed in this master thesis project (which is then presented in Section 3.2.2).

3.2.1 General (static) airport gate assignment

The original *over–constrained airport gate assignment problem* was presented in [10, 11] and inspired by the work presented in [1].
The problem contains a set of n flights – F – and a set of m airport gates – G. Ding et al. minimized the number of flights assigned to gates that are not located at a terminal/pier, as well as the passenger walking distances between (btw.) two gates k and l ($w_{k,l}$) and btw. the gates and the entry/exit ($w_{0,k}, w_{k,0}$).

The number of connecting passengers from flight i to flight j is denoted as $f_{i,j}$, where $f_{i,i} = 0$. Passengers can only board a flight which departs later than the arrival time of the origin flight.
Therefore, two possible cases arrise:

flight i precedes flight j: if $f_{i,j} > 0$, then $f_{j,i} = 0$

flight j precedes flight i: if $f_{j,i} > 0$, then $f_{i,j} = 0$

A third possibility would exist in case the ground times of flights i and j are long enough to allow a mutual transfer of passengers. However, this circumstance does not seem to be included in the model.

The binary decision variable $x_{i,k}$ assumes the value 1 in case flight i is assigned to gate k and 0 otherwise.

The model proposed in [11] considers only gates located at a terminal where passengers can walk between the gates instead of driving with buses. Apron positions are represented by a dummy gate *m+1* and the terminal entry/exit is denoted as gate *0*. In case the dummy gate is part of the set of gates G, the new set is denoted as G'.

$$\text{Minimize:} \quad \sum_{i \in F} x_{i,m+1} \tag{1}$$

$$\text{Minimize:} \quad \sum_{i \in F} \sum_{j \in F} \sum_{k \in G'} \sum_{l \in G'} f_{i,j} w_{k,l} x_{i,k} x_{j,l} + \sum_{i \in F} \sum_{k \in G'} f_{0,i} w_{0,k} x_{i,k} + \sum_{i \in F} \sum_{k \in G'} f_{i,0} w_{k,0} x_{i,k} \tag{2}$$

subject to:

$$\sum_{k \in G'} x_{i,k} = 1 \qquad \forall\, i \in F \tag{3}$$

$$x_{i,k} x_{j,k} (t_j^D - t_i^A)(t_i^D - t_j^A) \leq 0 \qquad \forall\, i \in F,\ k \in G \tag{4}$$

$$x_{i,k} \in [0,1] \qquad \forall\, i \in F,\ k \in G' \tag{5}$$

Ding et al. [10] used a multi objective approach which contains two parts:
Objective (1) minimizes the number of flights assigned to the apron (*m+1*), whereas the second objective minimizes the total passenger walking distance. The first part of Objective (2) refers to the walking distance of transfer passengers, the second part to departing passengers and the third part to arriving passengers.

This mathematical model is subject to following constraints:
Constraint (3) ensures that flight i can only be assigned to exactly one gate.
In case flights i and j are assigned to the same gate k ($x_{i,k}$=$x_{j,k}$=1), the departure time of the preceding aircraft has to be smaller than or equal to the arrival

time of the succeeding aircraft. This avoidance of overlapping flights at the same gate is indicated by Constraint (4) and does not include the dummy gate (*m+1*). Furthermore, the nonlinear constraint has to be linearized if required.
Constraint (5) indicates that $x_{i,k}$ is binary [12].

3.2.2 Robust approach for the airport gate assignment with airline preferences

The model described in this section is an extension of the *over–constrained airport gate assignment problem* which was presented in the previous section. Basic constraints – like time overlapping and decision variable binarity – are taken from the underlying model.

The three–part objective function was combined into a single aggregated objective function (6) via the weighted–sum method:

$$\text{Maximize:} \qquad \alpha_1 z_1 + \alpha_2 z_2 + \alpha_3 z_3 \tag{6}$$

where:

$$z_1 := \sum_{k \in G} \sum_{i \in F} x_{i,k} u_{i,k} \tag{7}$$

$$z_2 := \sum_{k \in G} \left[\left(t_k^{LD} - t_k^{FA} \right) - \sum_{i \in F} x_{i,k} \left(t_i^{D} - t_i^{A} \right) \right] \tag{8}$$

$$z_3 := g^{min} \tag{9}$$

$$\text{and} \qquad \alpha_1,\ \alpha_2,\ \alpha_3 \ \geq \ 0 \tag{10}$$

subject to:

$$\sum_{k \in G(i)} x_{i,k} = 1 \qquad \forall\, i \in F \tag{11}$$

$$\sum_{k \notin G(i)} x_{i,k} = 0 \qquad \forall\, i \in F \tag{12}$$

$$t_i^D + t_k^{\text{setup}} \leq t_j^A + M(2 - x_{i,k} - x_{j,k}) \qquad \forall\, k \in G,\ \ i,j \in F' \tag{13}$$

$$t_i^D + t_k^{\text{setup}} \leq t_j^A + M(2 - x_{i,k} - x_{j,l} + p_{i,j} + p_{j,i}) \qquad \begin{array}{l}\forall\, k,l \in G,\ i,j \in F \\ \text{where } s_{k,l} = 1\end{array} \tag{14}$$

$$t_k^{FA} \leq x_{i,k}\,(t_i^A - t_k^{\text{setup}}) + M(1 - x_{i,k}^{FF}) \qquad \forall\, k \in G,\ i \in F \tag{15}$$

$$\sum_{i \in F} x_{i,k}^{FF} \leq 1 \qquad \forall\, k \in G \tag{16}$$

$$t_k^{FA} = \sum_{i \in F} x_{i,k}^{FF}(t_i^A - t_k^{setup}) \qquad \forall\, k \in G \tag{17}$$

$$t_k^{LD} \geq x_{i,k}\, t_i^D - M(1 - x_{i,k}^{LF}) \qquad \forall\, k \in G,\ i \in F \tag{18}$$

$$\sum_{i \in F} x_{i,k}^{LF} \leq 1 \qquad \forall\, k \in G \tag{19}$$

$$t_k^{LD} = \sum_{i \in F} x_{i,k}^{LF}\ t_i^D \qquad \forall\, k \in G \tag{20}$$

$$g^{min} \leq t_j^A - t_i^D + t_k^{\text{setup}} + M(2 - x_{i,k} - x_{j,k}) \qquad \forall\, k \in G,\ \ i,j \in F' \tag{21}$$

$$0 \ \leq\ t_k^{FA} \leq\ 24\frac{60}{t^{IV}} - 1 \qquad \forall\, k \in G \tag{22}$$

$$0 \ \leq\ t_k^{LD} \leq\ 24\frac{60}{t^{IV}} - 1 \qquad \forall\, k \in G \tag{23}$$

$$x_{i,k} \ \in\ [0,1] \qquad \forall\, k \in G,\ i \in F \tag{24}$$

$$x_{i,k}^{FF} \ \in\ [0,1] \qquad \forall\, k \in G,\ i \in F \tag{25}$$

$$x_{i,k}^{LF} \ \in\ [0,1] \qquad \forall\, k \in G,\ i \in F \tag{26}$$

$$g^{min} \ \geq\ 0 \tag{27}$$

The first part of the aggregated objective function (Objective (7)) represents the airline preferences. In case flight i is assigned to gate k ($x_{i,k}$=1), the preference value $u_{i,k}$ is added to the objective value. The selection of the preference value scale results in many small MIP trade–off decisions, since more flights have to be assigned to their most preferred gate ($u_{i,k}$=100) in case one flight is assigned to an unwanted gate ($u_{i,k}$= –1000).

The second part of the objective function (Objective (8)) increases the robustness of the scheduling. A schedule is robust if the time between two consecutive flights at the same gate is long enough to be unaffected in case of time window shifts (e.g. flight earliness/tardiness). This objective is particularly important during daily planning, where a schedule is created for a time period which lasts several hours.
In the MIP, this effect is achieved by the maximization of all idle times between two consecutive flights at all gates.

Figure 6 shows an assignment of 4 flights to 2 gates. Although a gap of 30 minutes between two flights is already acceptable, the schedule can still be improved.

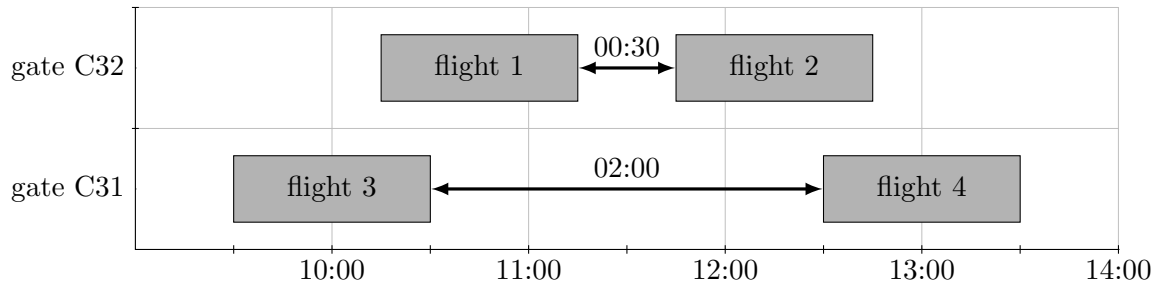


Figure 6: Example for a non–robust assignment (own graphic based on [9])

Figure 7 shows such an improved assignment where the smallest gap at all gates was increased from 30 minutes to 1 hour and 15 minutes by a simple exchange move between flight 2 and 4. The total gate idle time sum remains 2 hours and

30 minutes, but the smallest gap was increased, which makes the schedule more robust.

In general terms, the third part of the objective function increases the robustness by maximizing the smallest gap between any two consecutive flights at all gates.

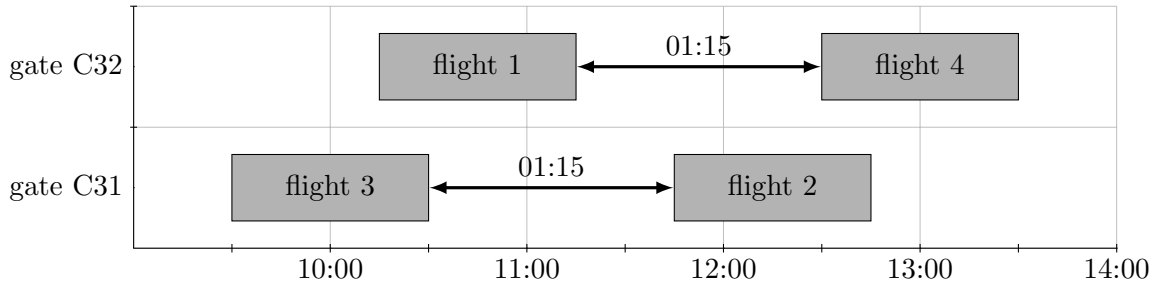


Figure 7: Example for a robust assignment (own graphic based on [9])

The mixed integer program is subject to the following constraints:

Constraints (11) and (12) ensure that flight i is assigned to exactly one gate. However, compared to Constraint (3) a different approach was used. Gates which are not possible for flight i, because of wingspan, length and gate characteristics, are eliminated from the assignment process in advance.

In case flight i and j are both assigned to gate k, the gap between the departure of flight i and the arrival of flight j has to be greater than or equal to the gate setup time t_k^{setup}. This is ensured by Constraint (13) and it guarantees that at most one aircraft can be parked at gate k at the same time ($p_{i,j} + p_{j,i} = 1$).

Constraint (14) embeds the shadow restrictions into the MIP. In case a shadow restriction exists between two gates ($s_{k,l}$=1), only one gate can be occupied at the same time. This static restriction is independent from the size of the aircraft parked at the affected gate.

Constraints (15)–(17) are used to calculate the arrival time of the first flight at every gate. Each gate can have a maximum of one first flight (Constraint (16)). In case there is at least one flight at the gate, t_k^{FA} assumes the smallest arrival time of all flights assigned to gate k minus the gate setup time (Constraints (15) and (17)).

Constraints (18)–(20) calculate the departure time of the last flight at each gate and are composed in a very similar way as Constraints (15)–(17).

Constraint (21) ensures that g^{min} is smaller than or equal to the smallest gap between any two consecutive flights i and j (assigned both to the same gate k). Since g^{min} is maximized in the third part of the aggregated objective function (9) it assumes the smallest gap at all gates.

Constraints (22)–(27) describe the ranges of the decision variables.
The first two constraints define the ranges of the first arrival time as well as the last departure time at each gate (e.g. if t^{IV}=5 then the range equals 00:00 – 23:55).

The last four constraints ensure the binarity and the non–negativity of the remaining decision variables.

4 Solution method

The *robust approach for the airport gate assignment problem with airline preferences (RAGA–AP)* MIP was solved using CPLEX, a commercial solver developed by IBM. Since exact algorithms are rather time consuming, a metaheuristic was adapted to the problem. The main goal of the metaheuristic is an attainment of the good solution for the problem set, but with a much shorter runtime than the exact algorithm.

At first, this chapter describes the metaheuristic *Large Neighborhood Search* in detail. In a second step, the implemented construction heuristic as well as the destroy and repair operators are presented.

4.1 Large Neighborhood Search

Large Neighborhood Search (LNS) is a metaheuristic which does not provide a guarantee to find the optimal solution. However, compared to a commercial MIP–solver, a metaheuristic needs much shorter runtimes.

LNS defines solution neighborhoods by the use of destroy and repair operators which ruin and recreate an incumbent solution x.
Destroy operators often involve stochastic elements to increase diversification. In case of RAGA–AP, this is achieved by the implementation of the *roulette–wheel selection*[7].

The selection of the *degree of destruction* is considered to be very important since it defines the number of unique neighborhoods. A small percentage might lead to problems during the exploration of the search space, whereas a high *degree of destruction* might lead to similar neighborhoods and a repeated optimization.

[7] A more detailed explanation of the simple *roulette–wheel selection* can be found in [18].

If we consider a CVRP with 100 customers and a *degree of destruction* of 15%, we obtain 2.5×10^{17} different selection possibilities. Combined with different repair operators, a high number of iterations is required to scan all possible neighborhoods. Different approaches on the selection of the *degree of destruction* can be found in the literature. These approaches range from a random selection during each iteration to a stepwise increase after a certain iteration number. We choose a fixed *degree of destruction* for all iterations during parameter tuning.

The implementation of repair operators allows much freedom and can consist of heuristics and/or optimal solution techniques. Latter achieve better results but also result in higher computation times and a limited amount of diversification [21].

Algorithm 1 Large Neighborhood Search (source: [21])

```
1: input: a feasible solution x
2: x^b = x;
3: repeat
4:    x^t = r(d(x));
5:    if accept(x^t,x) then
6:       x = x^t;
7:    end if
8:    if c(x^t) > c(x^b) then
9:       x^b = x^t
10:   end if
11: until stop criterion is met
12: return x^b
```

The pseudocode for Large Neighborhood Search is denoted in Algorithm 1.
A construction heuristic creates a feasible solution x, which provides the input for the LNS algorithm. x^b represents the best solution found so far and x^t is an incumbent solution which is subject to an acceptance decision. At the beginning, x^b is initialized with the starting solution.

During every iteration, an incumbent solution x^t is created by destroying and repairing the current solution x using the implemented operators. The function $accept(x^t, x)$ decides if x^t is a feasible solution and whether it is accepted as the new current solution x. In our case, this is done via a threshold–accepting of 98%, which allows 2% inferior solutions. In case a better solution was found, x^b is updated accordingly.
This ruin and recreate process is continued until the stop criterion is met. In case of the RAGA–AP, the algorithm terminates after a predefined number of iterations. At the end, the best solution is returned.

4.2 Proposed operators

This section describes the implemented construction heuristic as well as the destroy and repair operators of LNS in more detail.

4.2.1 Construction heuristic

The main purpose of a construction heuristic is the creation of a feasible starting solution where all flights are assigned and no aircraft remains ungated.

For every flight i, a list of possible gates – according to wingspan, length and gate characteristics – is created and sorted according to their preference values. A flight is chosen and assigned to an unoccupied gate with the highest preference value.
In order to avoid infeasible solutions, flights with a smaller number of possible gates have a higher probability to be chosen first.
In case the assignment process leads to such an infeasible solution, the schedule gets deleted and the algorithm starts again.

4.2.2 Destroy operators

Destroy operators remove n_r flights from a feasible solution and add them to a set of removed flights R. The selection of destroy operators for the Large Neighborhood Search tries to cover three different approaches of destruction and was inspired by the work presented in [20].

Destroy worst – D_w

The first operator selects n_r flights according to their current preference values $u_{i,k}$. If a flight is assigned to an undesired gate, a high probability ensures that it will be removed from the schedule. The weight that flight i is selected is defined by

$$\max_{i \in F,\, k \in G} \{u_{i,k}\} - u_{i,k} + 1 \tag{28}$$

where $\max_{i \in F,\, k \in G} \{u_{i,k}\}$ represents the highest of all assigned preference values.

e.g.:

bad assignment:	$100 - (-1000) + 1 = 1101$	high chance to be removed
good assignment:	$100 - 100 + 1 = 1$	low chance to be removed

A good performance of this operator is expected especially with the large 24–hour instance where not every flight can be assigned to its most preferred gate.

Destroy similar – D_s

The second destroy operator consists of three parts and selects n_r flights according to their time–window similarity. The motivation behind this operator lies in

the removal of an unfavorable candidate as well as possible substitutes featuring similar time windows.

The first flight to be removed is one which is affecting the smallest gap g^{min}. The probability for a flight to be removed is calculated as follows:

$$g^{max} - \min_{i \in F,\, k \in G} \{g_{i,k}^{prev},\ g_{i,k}^{succ}\} + 1 \tag{29}$$

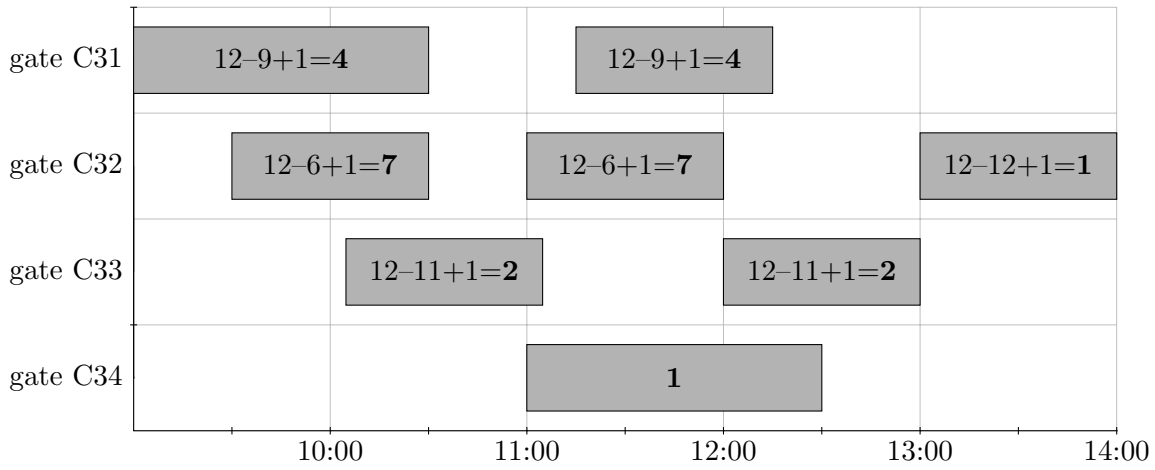


Figure 8: LNS operator *destroy worst* – first remove

Figure 8 shows the selection of the first flight. In this example, g^{max} is 1 hour (12 time units) and n_r equals 5. The flight assigned to gate C34 and the third flight at gate C32 have the lowest probability values since they contribute to the robustness the most. $\min_{i \in F,\, k \in G} \{g_{i,k}^{prev},\ g_{i,k}^{succ}\}$ returns the smallest gap of flight i to the previous flight ($g_{i,k}^{prev}$) or to the succeeding flight ($g_{i,k}^{succ}$) at gate k. In case flight i is the first or the last flight at the gate, it returns the gap to its sole neighbor; in case flight i has no neighbor the weight assumes the value 1.
The smallest gap g^{min} occurs between the first two flights at gate C32, hence the high values.

In the second step, and once the first flight is removed, $n_r - 1 = 4$ flights are selected according to their similarity to the first.
In this example, the second flight at gate C31 was selected first. Figure 9 shows the percentage of overlapping time with the first removed flight. The higher the overlapping value, the higher the probability that this flight will be removed in this step.
In our case, five flights shall be removed from the assignment in total, but only three flights overlap the ground time of the first removed aircraft. Therefore, a third step is implemented where flights with the closest time windows to the first remove are selected. In this example, this would be the first flight at gate C33.

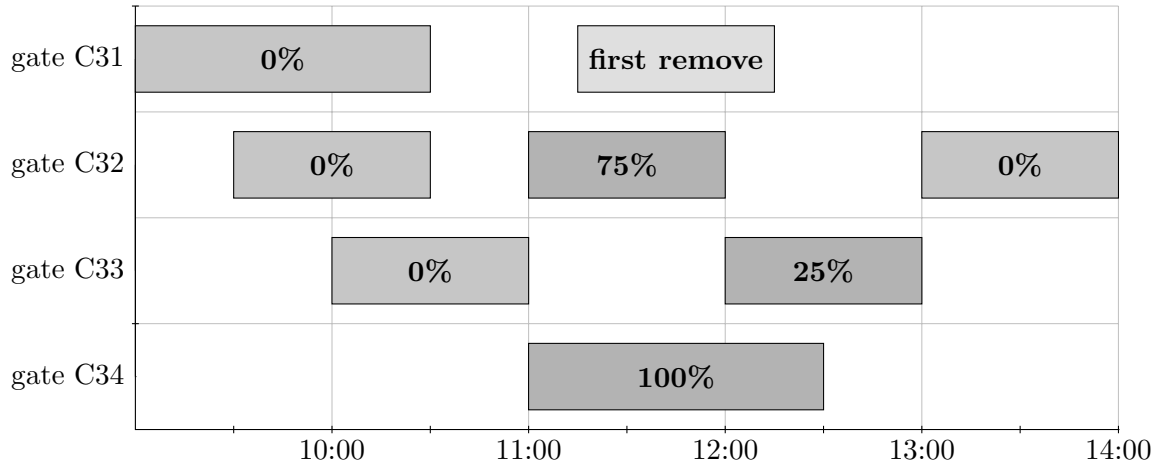


Figure 9: LNS operator *destroy worst* – n_r–1 removes

Destroy random – D_r

The last destroy operator is the most simple one. n_r flights are selected and removed from the feasible solution with the same probability.
A random removal increases the effect of diversification, since flights are not removed by an intelligent algorithm and also good assignments have the same probability to be selected [20].

4.2.3 Repair operators

The basic idea of repair operators is the reestablishment of a schedule's feasibility after a partial removal of flights. This applies if all flights are reassigned to gates and no constraint is violated. Inspired by the work presented in [20], two repair operators were implemented.

Before we take a closer look at the operators, the function *calc_objval_change(i_r,k)* is introduced. If a removed flight i_r is inserted into the sequence of gate $k \in G(i_r)$, the function calculates the change of the gate's objective value, denoted as $\Delta c_{i_r,k}$. The changes in a gate's total idle time sum t_k^{it} and its total preference value sum p_k is easy to calculate.

In case of g^{min}, a different approach had to be chosen, since the smallest gap at a gate (g_k^{min}) is only part of the aggregated objective function if it is the smallest gap at all gates ($g_k^{min} = g^{min}$).

However, it is considered quite important to include g_k^{min} into the calculation of $\Delta c_{i_r,k}$. Instead of adding g_k^{min} to $\Delta c_{i_r,k}$ directly, we take the deviation g_k^{dev} of the smallest gap at a gate from the smallest gap at all gates ($g_k^{dev} = g_k^{min} - g^{min}$).
In case g_k^{dev} is positive, the insertion of flight i_r into the sequence of gate k is desired; in case g_k^{dev} assumes a negative value, the assignment is not favored.

To sum up, the function *calc_objval_change(i_r,k)* is calculated as follows:

$$\Delta c_{i_r,k} = \alpha_1 p_k + \alpha_2 t_k^{it} + \alpha_3 g_k^{dev} \qquad \forall i_r \in R,\ k \in G(i_r) \tag{30}$$

Repair greedy_best $- R_g$

As its name implies, the *greedy_best* operator chooses the best insert position for flight i_r, regardless of the opportunity costs.

Based on the matrix containing all $i_r \in R \ \times \ k \in G(i_r)$ combinations calculated by the function *calc_objval_change(i_r,k)*, the repair operator *greedy_best* inserts a removed flight i_r to the gate with the highest $\Delta c_{i_r,k}$ value.

At first, the algorithm chooses a flight according to the number of possible gates to avoid infeasible solutions. Once flight i_r is selected, it is assigned to the gate with the best (greedy) impact on the objective value change:

$$(i_r,\ k) := \arg \max_{i_r \in R,\ k \in G(i_r)} \Delta c_{i_r,k} \tag{31}$$

In case the best gate is already occupied, i_r is assigned to the second best gate etc. After the assignment, flight i_r is removed from the set of removed flights R as well as the objective value change matrix and the $\Delta c_{i_r,k}$ values are updated for all remaining gates. The process is continued until all n_r flights are reassigned.

Repair regret_best $- R_r$

The second repair operator does not only assign a removed flight i_r to its best gate automatically, but it also considers the opportunity costs to the second best gate. As with the *greedy_best* operator, the *regret_best* operator first chooses a flight i_r according to its number of possible gates.

In a second step, the algorithm inserts flight i_r to a gate where the opportunity costs between the best and the second best gate are the highest:

$$(i_r,\ k) := \arg \max_{i_r \in R,\ k \in G(i_r)} (\Delta c^1_{i_r,k} - \Delta c^2_{i_r,k}) \tag{32}$$

The algorithm continues until all n_r flights are reassigned.

5 Computational experiments

This section presents the obtained computational results in detail. First, the input data as well as the parameter settings used for the model are described and an overview about the parameter tuning procedures is given. Later, Section 5.3 presents the main results of the CPLEX/LNS calculations.

All calculations were done on a LINUX cluster consisting of 8–core nodes of Intel® Xeon® CPU X5550 processors clocked at 2.67 GHz each and a shared memory of 24 GB. LNS was implemented in object–oriented C++ and the exact branch–and–cut algorithm for the MIP was computed by IBM ILOG CPLEX optimizer (version 12.1) embedded in a C++ concert technology.

5.1 Data sets

This section describes the source and configuration of all input data sets used in the model and provides information about the structure of the test instances. Furthermore, a description of the developed instance generator is given.

5.1.1 Input data

The model is solved for instances which are completely based on real life data. Besides the flight data provided by the Austrian Aviation Safety Authority *Austro Control GmbH (ACG)*, additional information about minimum ground times was supplied by the ground operations departments of Austrian Airlines (AUA) and Niki Luftfahrt GmbH (NLY). Data regarding towing procedures and turnaround splits was kindly provided by VIE movement control.

Flight data

The required flight data including flight numbers, flight callsigns and aircraft registrations of all in– and outbound flights (passenger, cargo and general aviation) handled at VIE on Tuesday July 12th 2011 was provided by ACG.

In general, VIE's summer schedule is busier than the winter schedule due to an increased demand in flights to holiday destinations. School summer holidays started on July 9th 2011 in six out of nine Austrian provinces, therefore it is assumed to be a flight day with a slightly higher workload than usual.
Additional information about departure and arrival airports regarding Schengen information and ICAO codes, as well as scheduled departure and arrival times were retrieved from http://data.flight24.com and http://www.viennaairport.com.
General aviation flights which are handled by VAH via the general aviation centers were removed from the instance since non–scheduled flights are assigned to gates manually by VAH.
For all aircraft of AUA and NLY based at VIE, the data had to be slightly modified since the schedule (flight–to–aircraft assignments) provided by ACG was based on actual times of arrival/departure. However, the model presented in Section 3.2.2 uses (estimated) scheduled times, not actual times.

In case the ground time of a turnaround is below its minimum, a manual aircraft change[8] was performed to ensure enough ground time for the handling agents.
All residual in– and outbound flights were combined into several aircraft turn–arounds (the period between arrival and departure at an airport gate) according to their aircraft registrations and flight times. After the data cleansing, the instance contains 401 aircraft turnarounds of passenger and cargo flights.

[8] **aircraft change:** a different aircraft of the same company and aircraft type is assigned to perform flight i: e.g. aircraft with registration OE-LDA is inbound Vienna and has a delay of 1 hour. The subsequent flight of OE-LDA (OS121 to Frankfurt) is assigned to a different aircraft (OE-LDG, also an Airbus A319) to reduce the delay of flight OS121.

Gate data and shadow restrictions

All relevant information about pier and apron gates at Vienna Intl. Airport is described in the "Vienna civil airfield - terms of use" available at [14]. This includes specifications about maximum aircraft dimensions and gate characteristics (Schengen, Non-Schengen, pushback necessary,...).

The model considers all gates except parking positions located in the maintenance area and at the general aviation centers east/west. Furthermore, parking positions B97–B99 (located at the eastern end of the apron) are not implemented, since they are only used as intermediate/holding positions during de–icing operations.
Information about shadow restrictions and overlapping gates can be found in the AIP Austria[9] or in the Appendix on page 57.

In total we consider 115 gates and 50 shadow restrictions for all our instances, which resemble the actual layout of VIE.

Aircraft data

Information about all aircraft types operating at VIE was derived from the *Eurocontrol Aircraft Performance Database v2.0* which is available online[10].
Among this are specifications about ICAO and IATA codes, wingspan, length, wake turbulence category and engine types (e.g. turboprop aircrafts should not be assigned to pier gates). The instance contains more than 150 aircraft types.

[9] AIP Austria LOWW AD 2.24-1-2 – *Aircraft Parking/Docking Chart - ICAO*
[10] http://contentzone.eurocontrol.int/aircraftperformance/, last accessed: June 19th 2012

Preference data

The instance for the airline preferences is based on personal observations during four years of employment at VIE and do not represent the official agreements between the airlines and Vienna Intl. Airport since they are subject to secrecy. Additionally, two former VIE employees completed and double–checked the data. Preferences are represented by integer numbers ranging from –1000 to +100 depending on airline, aircraft type and gate block.

Based on these observations, preferences for more than 70 airlines operating at VIE were included.

Parameter settings

Parameter values concerning the input data (e.g. time interval, minimum ground time and gate setup time) were adjusted according to real–world situations. Other parameters concerning Large Neighborhood Search and CPLEX have been fixed up–front (e.g. CPLEX runtime limit and gap limit) or after a more detailed analysis (e.g. LNS *degree of destruction* and destroy–/repair–operator combinations).

The length of a time interval t^{IV} was fixed to 5 minutes (meaning one time unit equals 5 minutes) according to real–world flight data. Scheduled arrival and departure times are usually shown in 5 minute intervals, whereas actual times of arrival/departure are shown in 1 minute intervals. This approach follows the *multiple time–slot model* presented in [16].

Another variable that needed to be fixed was the *gate setup time* t_k^{setup}. In reality, this time is different for most gates since it depends very much on the gate characteristic and the location on the apron. It primarily depends on the necessity of pushback (taxi–out positions have shorter setup times). Furthermore, it depends on the location of the pushback area. In case of VIE, a pushback at gate

C41 needs more time than at gate F17, because the pushback area is located in a one–way taxilane and the succeeding aircraft has to wait until the taxilane to the respective gate is vacant. In the model, the gate setup time was fixed to 5 minutes for all gates.

The adjustment of the *objective function weights* turned out to be very sensitive and was subject to protracted tests. The main focus lies on the compliance with the airline preferences, but a good trade–off was needed to achieve a robust scheduling.

For the small instances, the alpha values were predefined as follows:

$$\alpha_1 = \frac{1}{n} \qquad \alpha_2 = \frac{1}{n} \qquad \alpha_3 = \frac{2}{n}$$

In case of the large 24–hour instance, the objective function weights assume following values:

$$\alpha_1 = \frac{1}{1.15\,m} \qquad \alpha_2 = \frac{1}{1.15\,n} \qquad \alpha_3 = \frac{1}{10}$$

The *CPLEX–runtime limit* was set to 80 hours and the *gap limit* to 1%.

More sensitive parameters used for LNS (e.g. *degree of destruction* and *destroy– and repair–operator combinations*) were optimized during parameter tuning.

5.1.2 24–hour VIE instance

Our largest instance at hand – covering a timespan of 24 hours on July 12$^{\text{th}}$ 2011 – contains 401 turnarounds in total. 52% are operated by the Austrian Airlines Group (Austrian, Lauda Air and Tyrolean), 16% by NIKI, 5% by Lufthansa, 2% by AirBerlin and 25% by other airlines. Out of these 401 turnarounds, 56.4% are Schengen, 21.4% are Non–Schengen and 22.2% are mixed flights (in– and outbound flight are not of the same characteristic).

394 flights (98%) are passenger flights and only 2% are cargo flights.
Regarding night stops, 48 aircraft (12%) are parked at Vienna Intl. Airport from the previous evening and 47 aircraft stay at the airport until the next day.

In total, 32 different aircraft types and 207 different aircraft (registrations) operated at VIE during that day, whereof 92 are hub–based (AUA and NLY).

Implementation of towing procedures

In real life, aircraft with long ground times are often towed to remote parking positions in case they are parked at a very desired gate block. Therefore, the gate is not occupied unnecessarily and can be used for subsequent arrivals.
At Vienna Intl. Airport, such desired gate blocks are C– (12 parking positions located at *Pier West*) and D–gates (9 parking positions located at *Pier East*, but only 8 gates can be used simultaneously). The new terminal *SKYLINK* was not yet in operation during this master thesis project and its parking positions (F01–F37 and F04–F36) were then treated as apron gates.

Discussions with airport managers showed that the splitting of turnarounds is a complex task and is therefore often done manually during tactical planning. Nevertheless, a simple splitting algorithm was implemented in the model to show the effectiveness of towing procedures at gate planning. Turnarounds are only split in case the preference values are highest for *Pier East/West*, meaning the selection of alternative gate blocks featuring only apron gates is less desired. The emerging *parking turnarounds* have preferences at A–, H– and K– parking positions, since these gate blocks are rarely used for passenger flights.

In order to adapt the splitting algorithm to real life situations, it is important to distinguish between three types of turnaround splits:

night stops from the previous evening

At VIE, the first outbound wave starts between 06:00 and 07:00. In case the departure time of a night stop is later than 08:00, the turnaround is split into a parking and a departure part. As a result, the aircraft is parked at a remote parking position during the night and towed to its final parking position before the aircraft departs to its destination. In doing so, enough time is taken into account at the departure gate to allow handling agents and airline crews sufficient time to prepare the aircraft for its departure.

night stops until the next day

In case an aircraft arrives later than 20:00 and stays until the next morning, the turnaround is split again into an arrival and a parking part. After the deboarding and the preparations for the night stop by the handling agents, the aircraft is towed to a remote parking position until the next day.

long turnarounds during the day

The third possibility for a turnaround split occurs in case an aircraft has an above–average ground time during the day. An example for this would be an Austrian Airlines Boeing 767–300 which arrives in the morning from North America and departs to Asia in the afternoon.

In this case, the turnaround is split into three parts: an arrival, a parking and a departure part.

After the application of the splitting algorithm, the instance size increased from 401 to 431 turnarounds.

5.1.3 Small instances

In order to test and configure LNS for the large 24–hour instance, 15 small instances were created by an *instance generator* which randomly creates instances based on the characteristics of the 24–hour schedule. These instances are divided into morning, afternoon and evening instances each containing 10, 15, 20, 25 and 35 flights. The reason for this time–wise segmentation is the heterogeneity of traffic composition during these time intervals.

morning instances (B06_E12_*)

The time period between 06:00 and 12:00 contains the morning outbound rush of overnight parkers (06:30–07:30) and the first traffic wave (08:00–11:00) with a peak of 53 parking aircraft (see Figure 3 on page 6 for further details). This includes the majority of Austrian Airlines longhaul flights, connections to the biggest European hubs and about 40% of this day's cargo flights.

afternoon instances (B12_E18_*)

The afternoon interval between 12:00 and 18:00 contains three traffic waves with peaks at around 12:15, 14:45 and 17:00. Compared to the morning interval, where only one peak exists and most of the time windows are arranged very similarly, the afternoon interval offers more combination possibilities of aircraft which are parked at the same gate. This also has an effect on the LNS solution quality, which will be discussed in Section 5.3.1.

evening instances (B18_E24_*)

The third time period (18:00 – 24:00) contains two inbound waves which peak at 19:50 and 23:15 and includes also night stops until the next morning as well as the remaining 60% of cargo flights. As we only consider a limited time horizon of 24 hours, the turnarounds are supposed to end at midnight.

Instance generator

Based on the input parameters (interval start time t_I^S, end time t_I^E and instance size n_I), the instance generator selects all flights where $t_I^S \leq t_i^A$ and $t_i^D \leq t_I^E$.
A main task of the tool is to generate instances whose characteristics resemble the ones present at the time interval under consideration. Hence the program determines the current workload and generates instances with similar workloads accordingly. This ensures that the traffic curve in the small instance develops the same way as in the large instance (see Figure 3) and that traffic peaks and lows are represented correctly.

The algorithm selects flights in three steps (airline, aircraft type and in the end a specific turnaround) until the instance size equals n_I:
At the beginning, an airline is picked randomly. The more flights an airline operates within the time period, the higher the chance of this airline to be selected. This ensures that if an airline operates 50% of flights during the defined time interval, about $\frac{n_I}{2}$ flights in the small instance are flights of this particular airline. In case the chosen airline operates only one flight a day to/from VIE, this flight is added directly to the instance. In every other case, the tool continues the selection process.
In a second step, the algorithm choses an aircraft type from the airline picked in step 1. More precisely, the instance generator selects an aircraft according to its wake turbulence category, since this represents more or less aircraft wingspan and length.
In the third and last step, the actual flight is chosen. The algorithm takes a look at the instance workload and picks a flight if the workload at that time (t_i^A until t_i^D) has not yet reached its upper limit.

5.2 Parameter tuning

The main objective of parameter tuning is an optimization of the LNS results as well as a reduction of the runtime. This section describes the performed steps applied to the used instances. At the beginning, the *degree of destruction* is determined and the *number of iterations* is fixed.
In a second step, all *destroy– and repair–operator combinations* were tested, which lead to 21 different combinations[11]. The abbreviations for this test are shown in Table 4.
During each test, the stated objective value or runtime is an average of 6 LNS runs.

Abbreviation	**Operator**
D_r	destroy random
D_w	destroy worst
D_s	destroy similar
R_g	repair greedy_best
R_r	repair regret_best

Table 4: LNS operator abbreviations

Small instances

First tests have shown that instances with 35 flights (B*_E*_F35) seemed to experience the most difficulties to reach good results. Therefore the parameter tuning for the small instances was conducted on those three instances.

Figure 10 shows the development of the objective function depending on different *degrees of destruction* (1%–70%). The number of iterations for this test was fixed

[11] LNS–operator combinations: $[D_r, D_w, D_s, D_{r,w}, D_{r,s}, D_{w,s}, D_{r,w,s}] \times [R_g, R_r, R_{g,r}]$
In case of a two or three operator combination, one is chosen randomly at the beginning of every iteration.

to 15.000 and the selection of the destroy and repair operators was a combination of all implemented operators ($D_{r,w,s}R_{g,r}$).

Figure 10: Parameter test small instances – degree of destruction

The curve peaks at a degree of 50% and flattens slightly afterwards. Given that the values between a degree of 45–70% do not allow a distinct selection, a second test is conducted to look at the development of the average runtime where the best solution is found.

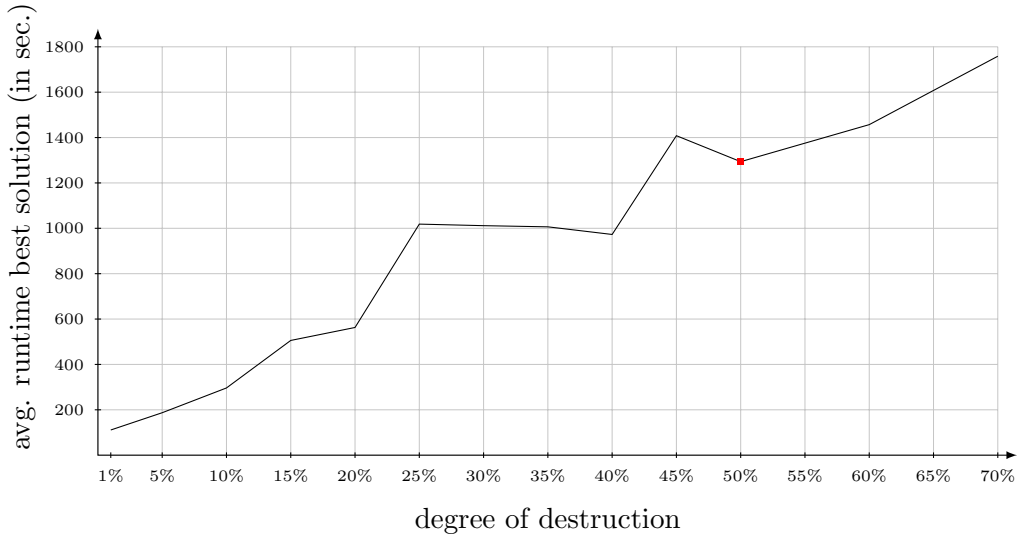


Figure 11: Parameter test small instances – runtime best solution found

Figure 11 shows the runtime where the best solution is found, depending on the *degree of destruction*.
The difficulty lies in the trade–off between objective value and runtime. Although the runtime at 40% is significantly lower, a degree of 50% is chosen, because the focus lies on the achievement of a higher objective value.

The final test selects the best *destroy– and repair–operator combination*. At first, all 21 possibilities are tested on the three 35–flight instances and the best methods are selected and tested on all 15 small instances.
Table 5 shows the three best operator combinations for the instances containing

Operators		**AOV**	**STDEV**
D_w	R_r	14.04	0.08
$D_{w,s}$	R_r	14.03	0.07
$D_{r,w,s}$	R_r	14.02	0.04

Table 5: Parameter test F35 instances – operator combinations

35 flights (B*_E*_F35) as well as the respective average objective value (AOV). Since the values are very close, the table also shows the average standard deviation (STDEV).
Particularly noticeable is the fact that the single use of the repair operator *regret_best* (R_r) achieves better results in every combination with the destroy operators than the single use of R_g or the combination of all repair operators ($R_{g,r}$).

Based on these results, the three best combinations mentioned above were tested on all 15 small instances. Taking the average objective value of 15 differently structured instances is not a very accurate comparison method. Therefore a comparison by ranks was introduced[12].

[12] For each instance, a rank between 1 and 3 is given to the three operator combinations depending on their respective objective value. Rank 1 is assigned to the combination with the highest value; 3 to the one with the lowest.

Operators		Rank	STDEV
D_w	R_r	1.33	0.024
$D_{w,s}$	R_r	1.53	0.027
$D_{r,w,s}$	R_r	**1.20**	**0.018**

Table 6: Parameter test small instances – operator combinations

Table 6 shows the average ranks of the proposed operator combinations for all small instances. The combination of all three destroy operators and the repair operator *regret_best* ($D_{r,w,s}$ R_r) emerge to be the best selection for the small instances. It achieves the highest rank and the smallest standard deviation compared to the other two combinations.

24–hour VIE instance

The parameter tuning for the large 24–hour instance follows the same steps as for the small instances. The number of flights to be removed during the destroy phase has been varied between 10 and 120 flights, at a stepsize of 10.

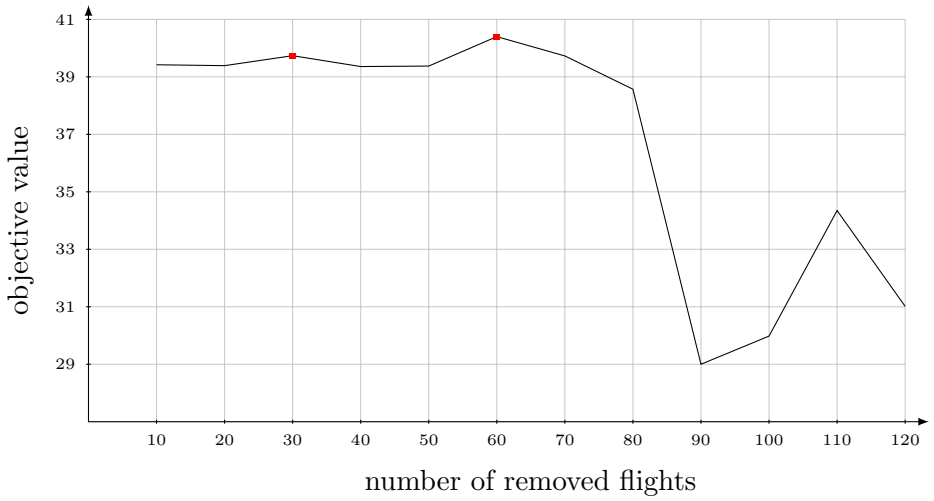


Figure 12: Parameter test VIE instance – degree of destruction

Figure 12 shows that, with a selection of 10 to 80 flights, good results can be achieved. Starting from 90 removed aircraft (*degree of destruction*: 22.4%), the average objective value drops significantly.

Now we want to take a closer look at the two peaks at 30 and 60 removed aircraft. Although in the previous section a higher objective value was preferred to a shorter runtime, this approach is questionable for this instance since the runtime range is approximately ten times higher. In Figure 13 we can see that a selection of 30 removed aircraft reduces the average runtime by almost 50%, whereas only 1.6% of the objective value have to be compromised for this trade–off.

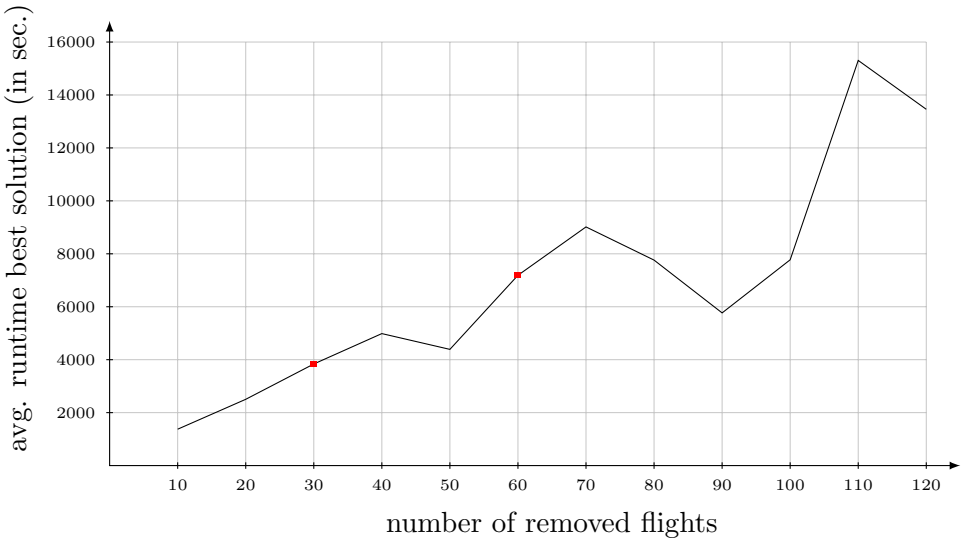


Figure 13: Parameter test VIE instance – runtime best solution found

The three best operator combinations for the large 24–hour instance are denoted in Table 7. It is quite surprising that the single use of the repair operator *greedy_best* achieved the best result. Nevertheless, the combination of all three destroy and all two repair operators ($D_{r,w,s}\, R_{g,r}$) is chosen, which results in a slightly worse average objective value (–1.4%), but shows a 27% smaller standard deviation (STDEV).

Operators	AOV	STDEV
$D_{r,w,s}\, R_g$	40.02	1.24
$D_{w,s}\ \ R_{g,r}$	39.42	1.62
$D_{r,w,s}\, R_{g,r}$	39.46	0.90

Table 7: Parameter test VIE instance – operator combinations

5.3 Experimental results

This section finally presents the computational results for the small instances as well as for the large 24–hour instance. At first, a recapitulation of the used parameters is given. Thereafter, the results are presented and analyzed.

5.3.1 Small instances

The final calculations for the 15 small instances were performed with a 50% *degree of destruction*, an *iteration limit* of 20.000, a mix of all three destroy operators combined with the repair operator *regret_best* ($D_{r,w,s}R_r$), a CPLEX *runtime limit* of 80 hours as well as a CPLEX *gap limit* of 1%.

Table 8 denotes the results of the calculations. The columns *Solution*, *SolTime* and *Gap* display the best solution found with CPLEX after 80 hours of computation, the corresponding time when the best solution was found (in seconds) as well as the CPLEX gap between the upper and lower bound.
The remaining columns show the best LNS solution found during 6 runs (*Best*), the average of these 6 runs (*Mean*) and the average solution time when the solution was found (*SolTime*). The last column presents the percental difference between the solutions obtained by CPLEX and LNS.

Apparently CPLEX struggles quite a bit during its calculations, which is reflected in very high *SolTimes* and an average *Gap* of 93.1%. For two instances (B06_E12_F35 and B18_E24_F35) no solutions are available due to memory issues.

	CPLEX/MIP			LNS			
Instance	Solution	SolTime	Gap	Best	Mean	SolTime	Δ CPLEX
B06_E12_F10	**15.40**	67	93.7%	**15.40**	**15.40**	1.85	0.00%
B06_E12_F15	**10.67**	67,180	94.2%	**10.67**	**10.67**	1.35	0.00%
B06_E12_F20	**12.45**	129,780	92.2%	**12.45**	**12.45**	6.15	0.00%
B06_E12_F25	**12.04**	130,192	91.3%	**12.04**	**12.04**	4.11	0.00%
B06_E12_F35	n/a	n/a	n/a	**10.94**	10.92	835.14	n/a
average		81,805	92.9%			169.72	0.00%
B12_E18_F10	**20.50**	80,647	92.9%	**20.50**	**20.50**	1.26	0.00%
B12_E18_F15	**20.67**	56,067	91.6%	**20.67**	**20.67**	11.56	0.00%
B12_E18_F20	**17.45**	136,154	91.4%	**17.45**	17.38	542.23	0.38%
B12_E18_F25	16.44	278,398	91.2%	**16.48**	**16.48**	156.35	-0.24%
B12_E18_F35	**17.06**	277,552	92.3%	16.89	16.81	1,087.01	1.42%
average		165,764	91.9%			359.68	0.31%
B18_E24_F10	**16.90**	704	95.5%	16.80	16.80	0.43	0.59%
B18_E24_F15	**14.40**	129,714	94.9%	**14.40**	**14.40**	87.36	0.00%
B18_E24_F20	15.10	130,613	93.7%	**15.45**	15.37	538.87	-1.77%
B18_E24_F25	**14.76**	273,856	93.8%	14.48	14.48	274.06	1.90%
B18_E24_F35	n/a	n/a	n/a	**14.46**	14.36	828.72	n/a
average		133,722	94.5%			345.89	0.18%

Table 8: Computational results – small instances

The runtimes required by LNS are highly competitive, especially when compared to those required by CPLEX. On average, the LNS runtimes assume values between 0.2% and 0.3% of the CPLEX runtimes, which prove the effectiveness of the implemented metaheuristic. Especially with instances of 10 and 15 flights, the best solution can be found in a very short period of time, which may be caused by the structure of the instances and its workload during the respective interval. In 12 out of 15 cases, LNS achieves to find the best solution at least during one

run and for 8 instances the best solution is found in every run. For instances B12_E18_F25 and B18_E24_F20 LNS finds better solutions than CPLEX; even at a fraction of the CPLEX runtime.

Furthermore it is noticeable that the number of traffic peaks within an instance time interval has a direct effect on the LNS solution quality. The lowest ΔCPLEX values are achieved for the morning instances and the highest for the afternoon instances. If we take a closer look at Figure 3 on page 6, we can see that the workload during the morning interval (06:00–12:00) peaks once at around 09:30. In case of the afternoon interval (12:00–18:00), three peaks at around 12:30, 14:45 and 17:00 are noticeable, which leads to much more combination possibilities for two or more flights at the same gate; hence the highest average Δ–value of 0.31%. During the evening interval (18:00–24:00), the traffic peaks twice at around 19:45 and 23:00, resulting in a Δ–value of 0.18%.
To sum up, the less traffic peaks an instance interval contains, the better the LNS results.

5.3.2 24–hour VIE instance

To make the model more realistic, towing procedures were introduced to improve the assignments at gate blocks where capacity limitations are most likely to occur. The main goal of such procedures is an increase in the number of assigned aircraft to a certain gate block with a consistent average idle time sum per gate block.

For the large 24–hour instance, CPLEX was not able to find a single solution within the imposed time limit of 80 hours, since the $n \times m$ matrix equals 401/431 flights $\times$ 115 gates which is simply too large to be solved optimally. Therefore, a comparison between an instance with (TOW) and one without towing procedures (NO TOW) is drawn. This should demonstrate the importance of turnaround splits during daily/tactical planning.

During parameter tuning, the *number of removed flights* was fixed to 30 aircraft, the *number of iterations* to 40.000 and the selected LNS operators are a combination of all destroy and all repair operators ($D_{r,w,s}R_{g,r}$).

Table 9 shows the computational results of the instance with and without towing procedures. In case of the instance without turnaround splits (NO TOW) on average 57 aircraft are assigned to 12 parking positions at *Pier West* (C–block – Schengen) and 32 aircraft are assigned to 8 parking positions at *Pier East* (D–block – Non–Schengen).

The instance including towing procedures (TOW) assigns on average 61 aircraft to the C–block and 40 aircraft to parking positions at the D–block. The average idle times at both piers remain constant with both instances.

	NO TOW	**TOW**	**improvement**
Best	37.42	40.59	
Mean	37.03	39.63	
SolTime	4553.78	4633.68	
avg. aircraft C	57	61	+4
avg. aircraft D	32	40	+8
avg. idletime C	02:40	02:40	00:00
avg. idletime D	02:35	02:30	– 00:05

Table 9: Computational results – VIE instance

Discussions with VIE airport managers showed that capacity bottlenecks appear especially at *Pier East* (D–block) where Non–Schengen, Non–EU and international flights are handled. A look at Table 9 confirms the advantages of an implementation of towing procedures, since more flights can be assigned to pier–gates while average idle times remain constant. In case of *Pier East* – where a maximum of 8 gates can be handled simultaneously – this means that an additional traffic wave (+8 flights) can be assigned to the pier gates located at the D–block, resulting in more passenger convenience while maintaining a robust schedule.

6 Conclusion

In this thesis, a new mixed integer program for airport gate assignment was developed and presented which is partly based on the *over–constrained airport gate assignment problem*. Previous research focused primarily on the minimization of passenger walking distances which is rather passenger/airline orientated.

The main objective of this model is a maximization of airline preferences as well as a creation of a robust schedule that is as insusceptible as possible against flight delays which arise during daily operations. In general, airport gate assignment can be divided into tactical, daily and seasonal planning. The new model belongs to daily planning.

At first, the problem was tested with 15 small instances and solved using a commercial MIP solver which resulted in good results, but very long computational times. Therefore, the metaheuristic *Large Neighborhood Search* was adapted to the problem with the objective of decreasing the runtime of the problem while maintaining good results.

Once the destroy and repair operators were optimized, the metaheuristic was applied to a large instance which simulates a 24–hour traffic period at Vienna International Airport. A version with and one without the implementation of towing procedures was presented and compared to highlight the importance of turnaround splits.

During these computations, *Large Neighborhood Search* proved to be a very effective metaheuristic for these kinds of combinatorial optimization problems.

An aggregation of different objective functions that combine passenger, airline and airport oriented goals offers many possibilities for further research. One particular possibility would be an assignment of flights to certain gate blocks and further optimization within this block regarding walking distances and schedule robustness.

References

[1] Babić, O., Teodorović, D., & Tošić, V. (1984). Aircraft stand assignment to minimize walking. *Journal of Transportation Engineering*, 110(1), 55–66.

[2] Bolat, A. (2000). Procedures for providing robust gate assignments for arriving aircrafts. *European Journal of Operational Research*, 120(1), 63–80.

[3] Braaksma, J. P. (1973). *A computerized design method for preliminary airport terminal space planning.* The Transport Group, Department of Civil Engineering, University of Waterloo, Waterloo, Ontario, Canada.

[4] Braaksma, J. P. (1977). Reducing walking distances at existing airports. *Airport forum*, 7.

[5] Braaksma, J. P. & Shortreed, J. H. (1975). Method for designing airport terminal concepts. *Transportation Engineering Journal of ASCE*, 101(2), 321–335.

[6] Chang, C. (1994). *Flight sequencing and gate assignment in airport hubs.* PhD thesis, University of Maryland at College Park.

[7] Delair (2011). SALLY – the airport resource management system. Available online at http://www.delair.eu/rms.html?file=tl_files/delair/pdf/sally_by_delair.pdf, last accessed: June 19th 2012.

[8] Diepen, G. (2008). *Column Generation Algorithms for Machine Scheduling and Integrated Airport Planning.* PhD thesis, Utrecht University. Available online at http://igitur-archive.library.uu.nl/dissertations/2008-0827-200530/diepen.pdf, last accessed: June 19th 2012.

[9] Diepen, G., van den Akker, M., & Hoogeveen, H. (2008). Integrated gate and bus assignment at Amsterdam airport Schiphol. In M. Fischetti & P. Widmayer (Eds.), *ATMOS 2008 - 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems* Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.

[10] Ding, H., Lim, A., Rodrigues, B., & Zhu, Y. (2004). New heuristics for over–constrained flight to gate assignments. *The Journal of the Operational Research Society*, 55(7), 760–768.

[11] Ding, H., Lim, A., Rodrigues, B., & Zhu, Y. (2005). The over–constrained airport gate assignment problem. *Computers & Operations Research*, 32(7), 1867–1880.

[12] Dorndorf, U., Drexl, A., Nikulin, Y., & Pesch, E. (2007). Flight gate scheduling: State–of–the–art and recent developments. *Omega*, 35(3), 326–334.

[13] Dorndorf, U., Jaehn, F., & Pesch, E. (2008). Modelling robust flight–gate scheduling as a clique partitioning problem. *Transportation Science*, 42, 292–301.

[14] Flughafen Wien (2010). Zivilflugplatz Benützungsbedingungen. Available online at http://viennaairport.com/jart/prj3/va/uploads/data-uploads/Passagier/Zivilflugplatz_Benuetzungsbedingungen.pdf, last accessed: June 19$^{\text{th}}$ 2012.

[15] Flughafen Wien (2012). FWAG (group) facts & figures. Available online at http://viennaairport.com/jart/prj3/va/main.jart?rel=en&content-id=1249344074274&reserve-mode=active, last accessed: June 19$^{\text{th}}$ 2012.

[16] Haghani, A. & Chen, M.-C. (1998). Optimizing gate assignments at airport terminals. *Transportation Research Part A: Policy and Practice*, 32(6), 437–454.

[17] Jaehn, F. (2008). *Robust Flight Gate Assignment (European University Studies Series V Economics and Management)*. Peter Lang GmbH.

[18] Lipowski, A. & Lipowska, D. (2012). Roulette–wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391(6), 2193–2196.

[19] Mangoubi, R. S. & Mathaisel, D. F. X. (1985). Optimizing gate assignments at airport terminals. *Transportation Science*, 19(2), 173–188.

[20] Pisinger, D. & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8), 2403–2435.

[21] Pisinger, D. & Ropke, S. (2010). Large neighborhood search. In M. Gendreau & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science* (pp. 399–419). Springer US.

[22] Şeker, M. & Noyan, N. (2012). Stochastic optimization models for the airport gate assignment problem. *Transportation Research Part E: Logistics and Transportation Review*, 48(2), 438–459.

[23] Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming* (pp. 417–431). London, UK: Springer-Verlag.

[24] Xu, L., Wang, F., & Xu, Z. (2011). A robust approach for the airport gate assignment. In J. J. Liu (Ed.), *Proceedings of IFSPA 2010*.

[25] Yan, S. & Tang, C.-H. (2007). A heuristic approach for airport gate assignments for stochastic flight delays. *European Journal of Operational Research*, 180(2), 547–567.

[26] Yan, S., Tang, C.-H., & Hou, Y.-Z. (2011). Airport gate reassignments considering deterministic and stochastic flight departure/arrival times. *Journal of Advanced Transportation*, 45(4), 304–320.

Abstract

The *airport gate assignment problem* is an important part of airport oriented research and gained more and more in importance during the last decades due to growing air traffic and subsequent tighter airline schedules.
The new deterministic model presented in this thesis focuses on the satisfaction of airline preferences as well as on the generation of a robust schedule which aims to be as insusceptible as possible against aircraft delays.
First applied to small instances derived from real–world data and then applied to a large instance containing a 24–hour traffic progression at Vienna International Airport, the implemented metaheuristic *Large Neighborhood Search* showed very good results regarding runtime and solution quality compared to benchmarks created by a *mixed integer program* solver.

Deutsche Zusammenfassung

Das *airport gate assignment problem* gewann in den letzten Jahrzehnten aufgrund der steigenden Dichte des Flugverkehrs und den daraus resultierenden dichter werdenden Flugplänen der Fluglinien immer mehr an Bedeutung. Das in dieser Magisterarbeit vorgestellte mathematische Modell konzentriert sich einerseits auf eine gute/bestmögliche Erfüllung der Präferenzen der Fluglinien für die gewünschten Parkpositionen, sowie andererseits auf die Erstellung einer Abstellzuordnung die so unanfällig wie möglich in Bezug auf Flugverspätungen ist. Die implementierte Metaheuristik wurde zuerst auf einige kleinere Testinstanzen und in weiterer Folge auf eine große Testinstanz, die einen 24 Stunden Flugbetrieb des Flughafen Wiens widerspiegelt, angewendet. Verglichen mit einem exakten Verfahren liefert die Metaheuristik, bei einem Bruchteil der benötigten Rechenzeit, sehr gute (teilweise auch bessere) Ergebnisse.

Curriculum Vitae

Personal Data

Name:	Gregor Gahbauer
Date of Birth:	May 12th 1986
Nationality:	Austria
E–Mail:	gregor@gahbauer.at

Education

2005 – 2012	University of Vienna – Bachelor and Master program field of study: Business Administration
1996 – 2004	Secondary school „Europagymnasium vom Guten Hirten Baumgartenberg“, Upper Austria
1992 – 1996	Primary school „St.Nikola an der Donau“, Upper Austria

Working Experience

03/2011 – 02/2012	Teaching Assistant at the Chair of Production and Operations Management, University of Vienna, Austria
07/2010 – 08/2010	Internship Ground Operations and Sales Management, Viennajet Bedarfsflug GmbH, Vienna, Austria
07/2006 – 10/2010	Aircraft Handling Services, Vienna Airport, Austria

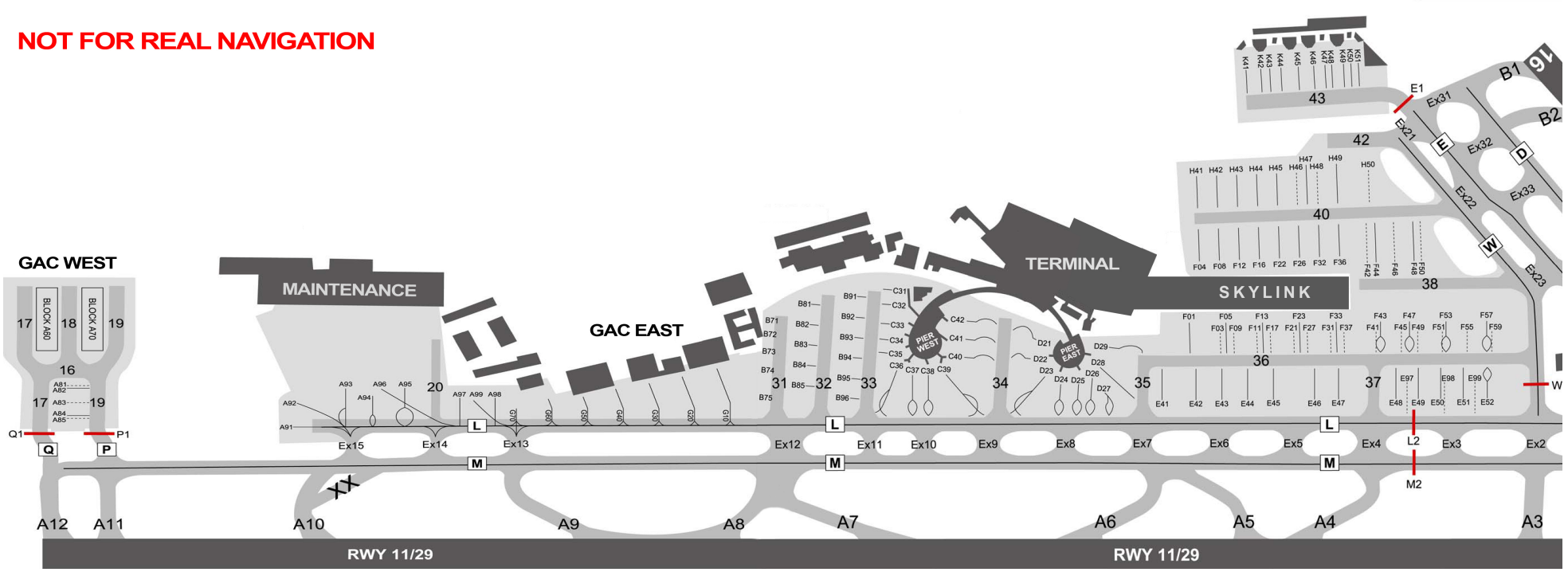
Additional Skills

German	mother tongue
English	language proficient
Spanish	fluent written and spoken
French	fluent written and spoken
Language stays	Cambridge (UK) 2000, Cannes (FR) 2003 and Australia 2005

IT skills	Good programming skills in C++, HTML, PHP and mySQL Solid knowledge in application of Microsoft Office including Excel Solver Basic knowledge in IBM CPLEX, ADOscore and Google AdWords
Pilot licenses	GPL, MiM and AFZ
Driving license	B

Vienna, June 2012

Appendix – VIE ground layout



source: VACC Austria (chart slightly modified) available online at www.vacc-austria.org