universität wien

DIPLOMARBEIT

Titel der Diplomarbeit

„Variable Neighborhood Search for the Static Aircraft Sequencing Problem“

Verfasser

Thomas Hermann

angestrebter akademischer Grad

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

Wien, 2011

Studienkennzahl lt. Studienblatt:	A 157
Studienrichtung lt. Studienblatt:	Diplomstudium Internationale Betriebswirtschaft
Betreuer:	O.Univ.-Prof. Dipl.-Ing. Dr. Richard F. Hartl

Eidesstattliche Erklärung

Ich erkläre an Eides statt, daß ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Wien, im Dezember 2011 Thomas Hermann

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Variables

P	Set of airplanes: $P = \{1, \dots n\}$
n	Number of airplanes: $\lvert P \rvert = n$
R	Set of runways: $R = \{1, \dots m\}$
m	Number of runways: $\lvert R \rvert = m$
E_i	Earliest landing/take-off time of airplane $i \in P$
T_i	Target (preferred) landing/take-off time of airplane $i \in P$
L_i	Latest landing/take-off time of airplane $i \in P$
B_i	Marginal penalty cost for airplane $i \in P$ for being before T_i: $B_i \geq 0$
A_i	Marginal penalty cost for airplane $i \in P$ for being after T_i: $A_i \geq 0$
S_{ij}	Minimum separation time between airplanes i and j, if airplane i lands/departs before airplane j on the same runway: $S_{ij} \geq 0 \; \forall i, j \in P, i \neq j$
s_{ij}	Minimum separation time between airplanes i and j, if airplane i lands/departs before airplane j on a different runway: $0 \leq s_{ij} \leq S_{ij} \; \forall i, j \in P, i \neq j$
t_i	Computed (scheduled) landing/take-off time of airplane $i \in P$: $t_i \geq 0$
b_i	Duration airplane $i \in P$ lands/departs before its target time T_i
a_i	Duration airplane $i \in P$ lands/departs after its target time T_i
z_{ir}	1: if airplane $i \in P$ lands/departs on runway $r \in R$, 0: otherwise
γ_{ij}	1: if airplanes i and j land/depart on the same runway, 0: otherwise, $\forall i, j \in P, i \neq j$
δ_{ij}	1: if airplane i lands/departs before airplane j, 0: otherwise, $\forall i, j \in P, i \neq j$

$s_{preordered}$	Sequence of pre-ordered airplanes
s_r	Sequence of airplanes assigned to runway $r \in R$: $s_r = x[r]$
x	Any solution; i.e. set of sequences $s_r, \forall r \in R$: $x = \{s_0, s_1, \dots s_m\}$
N_k	Set of neighborhood structures: $k = 1, \dots, k_{max}$
k	k^{th}-neighborhood
Z_{opt}	Evaluation value of optimal solution
Z_{best}	Evaluation value of best solution found by [15] in the course of their computational research (if optimal solution unknown)
Z_{init}	Evaluation value of initial solution
Z_{min}, Z_{max}	Minimum/Maximum evaluation value over 10 runs
Z_{avg}	Average evaluation value over 10 runs
t_{avg}	Average CPU-time over 10 runs in seconds, i.e. average time per run
$t_{avg_{mean}}$	Mean value of t_{avg} over all instances in seconds
T	Total CPU-time in seconds: comprises computation of initial solution and all replications performed
T_{mean}	Mean value of T over all instances in seconds
RPD_{min}	Relative Percentage Deviation of Z_{min} in relation to Z_{opt} or Z_{best}
$RPD_{min_{mean}}$	Mean value of RPD_{min} over all instances
G_{best}	Percentage Gap of best solution found by [6], [7] and [15] over 10 runs
G_{SW}	Percentage Gap of solution found by [20]

List of Abbreviations

ADP	Aircraft Departure Problem
ALP	Aircraft Landing Problem
ASP	Aircraft Sequencing Problem or Aircraft Scheduling Problem
ATC	Air Traffic Control
BA	Bionomic Algorithm
CTOT	Calculated Take-Off Time
CTSP-RT	Cumulative Travelling Salesman Problem with Ready Times
ICAO	International Civil Aviation Organization
JSS	Job Shop Scheduling
KTS	Knots (1 KTS = 1 Nautical Mile per hour = 1.852 km/h)
MIP	Mixed Integer Program
MILP	Mixed Integer Linear Program
MUC	International 3-letter code of Munich “Franz Josef Strauss”-Airport
SS	Scatter Search
SVNS	Multi-start Adaptive Variable Neighborhood Search with Taboo Memory
SW	Sliding Window Algorithm
TSPTW	Travelling Salesman Problem with Time Windows
VNS	Variable Neighborhood Search
VRPTW	Vehicle Routing Problem with Time Windows

1. Introduction

1.1 Motivation

The rapid growth of worldwide civil air traffic causes airspaces surrounding large airports and their respective runways to become more and more the bottlenecks in the air traffic system. For various reasons, expansion of airports infrastructures is not always a suitable alternative. Hence, efficient utilization of available capacities is of utmost importance. This is mainly achieved by what is known as runway operations scheduling, i.e. assigning each departing and arriving flight to an appropriate runway, computing a sequence for each runway and scheduling landing and take-off times for each airplane.

In Operations Research this topic is known as Aircraft Sequencing Problem or Aircraft Scheduling Problem (ASP). In literature the Aircraft Landing Problem (ALP) is studied predominantly. However, a number of papers also deal with the Aircraft Departure Problem (ADP). Basically these are special cases of the ASP where only landing or departing airplanes are considered respectively. Although very similar, vital differences between ALP and ADP exist. While the former usually only concentrates on immediate runway traffic, does the latter in general also consider the surrounding taxiway structure. Nevertheless, the models are basically the same and may also be applied to problems involving a mix of departures and arrivals.

In the last decades numerous exact and heuristic solution algorithms for the ASP were developed. Because of the wide solution space encountered when dealing with this type of problem, heuristics are of special interest. In this paper, Variable Neighborhood Search (VNS) will be applied as solution method. VNS is a modern, well-known metaheuristic proved to work excellently on various problems in the field of Operations Research. The ASP will be considered as a routing problem, and a VNS algorithm, basically designed for problems of transportation logistics, adopted accordingly. To the best of my knowledge this is not only the first application of VNS as proposed by [14] to solve the ASP, but also the first application of a metaheuristic to solve it in the context of a routing problem.

Both the single and the multiple runway cases are studied. The performance of the algorithm will be measured using benchmark sets provided by the OR-Library of J.E.

Beasley.[1] Additionally, the results obtained by the VNS will be compared to the results of recent most efficient heuristics. Finally, the algorithm will be applied to a real world scenario encountered at Munich Airport.

The scope of this thesis is limited to the static case, assuming complete information is given, i.e. all things are known in advance. Furthermore, it deals with the decision problem only. In other words, its focus lies solely on immediate runway traffic. It does neither cover the question how the computed solution will be achieved, nor whether it is achievable either. This kind of control problem is task of more fine grained models. Finally note that this paper only covers the basic version of the ASP. Various extensions as depicted in a later chapter are in fact not incorporated.

1.2 Outline

This thesis is organized as follows: Chapter 2 starts with a general overview of the characteristics of the ASP. Problem-specific terms are defined in a contextual sense and different viewpoints of various stakeholders are introduced. In the following, a detailed description of the problem, as considered in the underlying paper, is given and a mathematical formulation presented. Finally, similarities to routing problems are identified and the complexity of the problem is defined.

Chapter 3 initially introduces the scheme of the Basic VNS adopted. The rest of this chapter concentrates on a very detailed description of the various parts of the implemented algorithm.

Chapter 4 provides detailed results of computational experiments executed on a number of test problems. A comparative analysis compares the performance of the implemented algorithm with “state-of-the-art”-heuristics. Finally, an application of the metaheuristic to a real world scenario is described in detail.

Chapter 5 summarizes gained insights and gives an outlook of future research on the topic.

[1] See [4].

2. The Aircraft Sequencing Problem

2.1 Problem Overview

Airplanes inbound to an airport dynamically are assigned expected landing times by Air Traffic Control (ATC). These periodically updated estimated times of arrival are based on the planned routings and current speeds of the inbounds to the airport. Of course, the closer the flights to the airport the more accurate are the estimates.

In principle, for pre-planning purposes of the air traffic controller, the time an airplane arrives can be bounded by an earliest and a latest time, called the natural time window. The earliest time derives from flying at maximum speed straight to the airport. The latest time is influenced by remaining fuel available, limiting a flight´s capability to wait until it has to land. In fact, these time windows are hard constraints.

When entering the radar range of the respective air traffic approach control, arriving flights must be assigned to appropriate runways. In addition, sequences and schedules for each single runway must be computed on the basis of defined time windows. In this context, the awareness of the presence of so called wake turbulences is of utmost importance. Wake turbulences are wing tip-induced turbulences produced by every airplane that can be dangerous to its immediate successors. Therefore, officially published minimum separation requirements between airplanes must be adhered to at any time. In fact, these specific timely or spatially expressed sequence-dependent spaces are regulated in [12]. Briefly explained, a smaller aircraft behind a larger one needs more safety spacing than vice versa. Note that as long as separation requirements among immediately successive airplanes are fulfilled, successive separation is provided. To comply with the required complete separation, minimum separation rules among all pairs of airplanes must be observed.

Of course, similar definitions and restrictions, as described for arrivals, apply to departing flights. The earliest time departing traffic is able to take-off is mainly influenced by the time it is ready to leave the gate, and the taxi time required to the respective runway. In contrast, theoretically, there does not exist a latest take-off time for any departure, as fuel is not a factor when parked at the gate. But in practice, as arbitrary departure delays must be omitted, a latest time is implicitly defined anyway. In addition to these natural time windows, artificial time windows, so called Calculated Take-Off Times (CTOT), may be

allocated to outbound flights. These centrally coordinated time slots, aiming to avoid airspace congestion, override any defined natural time windows.

In the presence of multiple runways, one can distinguish between dependent and independent runway operations. This is primarily influenced by the physical structure of an airport. Geographical layout and spacing between runways are the major factors in determining whether runways are considered as being dependent or independent. The main concern here is again given to separation criteria. In other words, in the presence of dependent runways, operation on one runway influences operation on the dependent runway. This does not apply if runways can be considered as being independent. As one can imagine, dependent runway operations are in fact much more restrictive and complex to handle than independent operations are.

Multiple runways, whether they are dependent or independent, can be operated in a segregated, mixed or semi-mixed mode. Segregated mode means, one or more runways are solely dedicated to departures, while others to arrivals. In a mixed operation, all runways are available to both departing and arriving traffic. The semi-mixed mode can be considered as being in between.

Three parties are mainly involved in airport operations: ATC, the airport operator itself, and operating airlines. Each of them pursues different objectives. ATC might seek to maximize runway throughput, i.e. to maximize the number of flights served per unit of time, respectively minimize the time a set of flights is handled. Another common objective of ATC is the minimization of inbound and outbound delays. This is defined by the time an inbound flight is in the air and an outbound flight is on the ground respectively. The airport operator usually has in mind to observe punctuality concerning its own operative schedule (gate scheduling, etc.). The main concern of operating airlines of course is punctuality regarding their officially published schedules, which is completely ignored by ATC at present.

Note that mentioned objectives partially contradict themselves. Hence, defining an appropriate objective strongly depends on the viewpoint of the stakeholder. In fact, various objectives are incorporated in existing literature. Nevertheless, the “first-come first-serve”-methodology is still prevailing at many large international airports. In other words, the first airplane entering the radar range is the first to land. This may be the easiest way to manage air traffic, but in fact, it is far away from optimum of any stakeholder´s objective.

2.2 Problem Description

In this paper, as prevalent in recent literature, the “target time”-approach as introduced by [1] was put to practice. Following a number of authors, the basic version of the static ASP, defined as decision problem, will be considered.

In this context, a set of landing and/or departing airplanes P, with $|P| = n$, and a set of independent runways R, with $|R| = m$, are given. It is assumed that each runway $r \in R$ is operated in a mixed mode and each airplane $i \in P$ does not underlie any runway restriction.

With each airplane $i \in P$, a time window $[E_i, L_i]$ is associated, with E_i and L_i being the runway independent earliest and latest landing or take-off times. Bounded by each individual time window, a specific target landing or take-off time T_i, for $\forall i \in P$, with $E_i \leq T_i \leq L_i$, will be defined.

The target time T_i can be interpreted as an airplane´s preferred landing or take-off time. Mathematically formulated, the target time corresponds to the point in time, at which no additional costs for the respective airplane $i \in P$ incur. Note, by the term “additional” one means no costs apart from those existing anyway.

In contrast, for an airplane $i \in P$ being scheduled before or after its target time T_i, marginal additional costs, i.e. costs per unit of time, of $B_i \geq 0$ and $A_i \geq 0$ respectively, incur. For airplanes being in flight, earliness costs can be interpreted as additional fuel costs of speeding up, while tardiness costs are more related to maintenance costs, as the airplane remains in flight for a longer period of time. Figure 2.1 shows the run of the cost function for a single airplane $i \in P$.

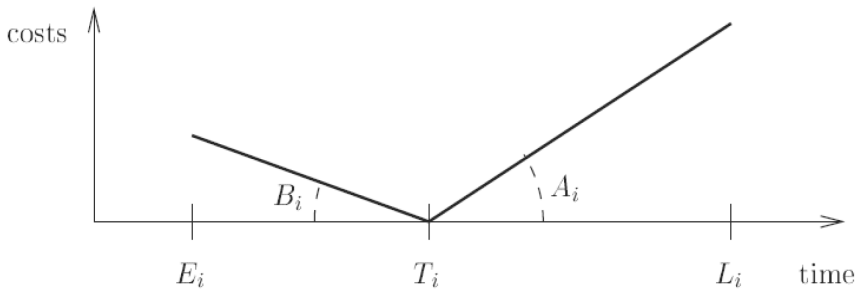

Figure 2.1: Run of the cost function for a single airplane.[2]

[2] Cf. [8] p. 156.

Note, a linear run of the cost function is assumed. Its positive slopes are determined by constants B_i and A_i, with no additional costs at target time T_i. Note further that the cost function must be concave, because: $B_i \geq 0$ and $A_i \geq 0$.

The objective considered in this paper is defined as the adherence to given individual target times T_i, for $\forall i \in P$, while earliness and tardiness will be penalized. Therefore, the objective function is represented by the total (additional) cost function (see Formula (1)).

$$total\ cost = \sum_{i=1}^{n}(B_i \cdot b_i + A_i \cdot a_i) \quad (1)$$

with:

$$b_i = max\{0, T_i - t_i\} \qquad \forall i \in P \quad (2)$$

$$a_i = max\{0, t_i - T_i\} \qquad \forall i \in P \quad (3)$$

In a contextual sense, the objective can be formulated as the minimization of the total weighted deviation from individually defined target times T_i, for $\forall i \in P$, with constants $B_i \geq 0$ and $A_i \geq 0$ being the weighting factors, by specifying an exact time $t_i \geq 0$ for each airplane $i \in P$ on a single runway $r \in R$, such that each airplane $i \in P$ lands or departs within its given time window $[E_i, L_i]$ with respect to minimum separation times $S_{ij} \geq 0$, for $\forall i, j \in P, i \neq j$, observed between all pairs of airplanes (i, j), for $\forall i, j \in P, i \neq j$, being assigned to the same runway $r \in R$.

An important point to consider in this context is the observance of complete separation, i.e. the fulfilment of separation criteria between all pairs of airplanes, and not only between its immediate successors. In fact, if the triangle inequality of the problem´s minimum separation matrix is valid, successive separation is sufficient to guarantee complete separation. Otherwise, this is not automatically the case. Recall, the triangle inequality is defined as:

$$S_{ik} \leq S_{ij} + S_{jk} \qquad \forall i, j, k \in P, k \neq i, j \neq i, k \quad (4)$$

with S_{ij}, S_{ik}, S_{jk} being the minimum separation times among successive airplanes *i*, *j* and *k* allocated to the same runway. In general, as long as the flow of air traffic is homogenous, i.e. airplanes are of similar size, or only departing or arriving flights considered, the separation matrix fulfils the triangle inequality. In contrast, in the presence of a

heterogeneous flow, i.e. a mix of departures and arrivals of airplanes belonging to different wake turbulence categories, the triangle inequality most probably does not hold. This can be easily demonstrated by considering airplanes i, j and k being assigned to the same runway, and sequenced as follows: i lands first, then departs j, finally lands k. If i belongs to wake turbulence category[3] "Heavy", and both j and k to category "Medium", the minimum separation times in a real world scenario would be: $S_{ij} = 50s$, $S_{jk} = 50s$, $S_{ik} = 120s$. It can easily be recognized, that in this case the triangle inequality does not hold. In other words, successive separation is not sufficient to ensure complete separation.

Finally note, as target times are solely related to airplanes, in this approach the minimization of total additional airplane costs is of major interest. It might seem that the adopted objective function reflects the objectives of operating airlines, but in fact this is not the whole truth, as can be demonstrated by the following example: From the viewpoint of an operating carrier, a tardy flight will speed up to enable its passengers matching their connecting flights. In this case, an airline´s preferred time for a given tardy flight to land will most probably be close to its earliest time, but for sure far away from the airplane´s target time. Nevertheless, the "target time"-concept adopted in this paper seems to be a good compromise between objectives of various stakeholders involved in airport operations.

2.3 Mathematical Formulation

For the sake of completeness and thorough understanding a mathematical formulation of the ASP, as considered in this thesis, will be presented.

In the vast majority of prevailing literature, mathematical models are only given for the ALP. In fact, what the models in general do is determining times dedicated to airplanes, subject to time window constraints and separation criteria. The models itself do not distinguish between arriving and departing flights. Hence, models developed for the ALP are in general also applicable to problems involving only departures, or a mix of departures and arrivals.

In this section a Mixed Integer Program (MIP) corresponding to [1], [8] and [15] is used to mathematically formulate the ASP in the context of the underlying thesis. For the purpose of better readability and consistency the notation was slightly modified.

[3] An airplane´s wake turbulence category depends on its Maximum Certified Take-Off Mass.

2.3.1 Objective Function

As already described in detail, the objective function represents the total cost of deviation from individual target times:

$$total\ cost = \sum_{i=1}^{n}(B_i \cdot b_i + A_i \cdot a_i) \quad (5)$$

with:

$$b_i = max\{0, T_i - t_i\} \qquad \forall i \in P \quad (6)$$

$$a_i = max\{0, t_i - T_i\} \qquad \forall i \in P \quad (7)$$

2.3.2 Single Runway Constraints

Time window constraints:	$E_i \leq t_i \leq L_i$	$\forall i \in P$	(8)
Sequence constraints:	$\delta_{ij} + \delta_{ji} = 1$	$\forall i, j \in P, i < j$	(9)
Separation constraints:	$t_i + S_{ij} \leq t_j + M \cdot \delta_{ji}$	$\forall i, j \in P, i \neq j$	(10)
Time-related constraints:	$t_i + b_i - a_i = T_i$	$\forall i \in P$	(11)
Binary variable:	$\delta_{ij} \in \{0,1\}$	$\forall i, j \in P, i \neq j$	(12)
Non-negativity:	$t_i, a_i, b_i \geq 0$	$\forall i \in P$	(13)

Constraint (8) ensures time window feasibility. It guarantees that all airplanes land or depart within their respective time windows.

Constraints (9) and (10) decide the runway sequence. The former determines that either airplane i or airplane j has to land or depart before the other, while the latter ensures compliance with complete separation. If $\delta_{ji} = 0$, i.e. airplane i is ahead of j, Constraint (10) will be considered, otherwise constant M makes it redundant by assigning a large value to the right hand side of the inequality.

On the one hand, M should be large enough to "disable" the constraint if applicable, on the other hand, to strengthen the LP-relaxation of the MIP, it is favorable to choose it as small as possible. Furthermore, it should consider airplanes involved. Therefore, an adequate choice of this constant seems to be:

$$M = L_i + S_{ij} - E_j \qquad \forall i, j \in P, i \neq j \qquad (14)$$

Constraint (11) relates b_i and a_i to scheduled time t_i and target time T_i. As long as the cost function is concave, which is true in the underlying model (recall: $B_i \geq 0$, $A_i \geq 0$), not both variables b_i and a_i will be non-zero in an optimal solution. Therefore, the deviation from target time T_i will be computed correctly.

Finally, Constraints (12) and (13) correspond to the binary and non-negativity constraints respectively.

2.3.3 Multiple Runway Constraints

Time window constraints:	$E_i \leq t_i \leq L_i$	$\forall i \in P$	(15)
Sequence constraints:	$\delta_{ij} + \delta_{ji} = 1$	$\forall i, j \in P, i < j$	(16)
Runway allocation:	$\gamma_{ij} = \gamma_{ji}$	$\forall i, j \in P, i \neq j$	(17)
Separation constraints:	$t_i + S_{ij}\gamma_{ij} + s_{ij}(1 - \gamma_{ij}) \leq t_j + M \cdot \delta_{ji}$	$\forall i, j \in P, i \neq j$	(18)
Multi-runway constraints:	$\sum_{r=1}^{m} z_{ir} = 1$	$\forall i \in P$	(19)
	$z_{ir} + z_{jr} - 1 \leq \gamma_{ij}$	$\forall i, j \in P, i < j$ $r \in R$	(20)
Time-related constraints:	$t_i + b_i - a_i = T_i$	$\forall i \in P$	(21)
Binary variables:	$\delta_{ij}, \gamma_{ij}, z_{ir} \in \{0,1\}$	$\forall i, j \in P, i \neq j$ $\forall r \in R$	(22)
Non-negativity:	$t_i, a_i, b_i \geq 0$	$\forall i \in P$	(23)

Note that time window, sequence, and time-related constraints are the same as for the single runway case. The same applies to the binary and non-negativity constraints.

Constraint (17) makes sure, if airplanes i and j use the same runway, so too must airplanes j and i.

Constraint (18) ensures compliance with complete separation. If $\delta_{ji} = 0$, i.e. airplane i is ahead of j, it ensures that a minimum separation time of S_{ij} or s_{ij} (depending on γ_{ij}, i.e. whether or not they use the same runway) between airplanes i and j is observed. Otherwise, constant M makes above expression redundant by assigning a large value to the right hand side. The choice of M is the same as for the single runway case. Note, as this paper only considers independent runways, the dependent minimum separation time variable $s_{ij} = 0$.

Constraints (19) and (20) assign airplanes to runways. The former ensures that each airplane is assigned to only one runway. The latter makes sure that if airplanes i and j use the same runway, the runways assigned to them are consistent.

2.3.4 Complete MIP-Formulation

The complete MIP-formulation consists in minimizing the total cost function (5) subject to Constraints (8) – (13) and (15) - (23) for the single and multiple runway cases respectively. Expressions (6), (7) and (14) are incorporated appropriately.

There would be a number of possible extensions to the model formulated (e.g. runway dependent earliest, target and latest times, runway dependent separation times, runway restrictions, etc). As this paper only covers the basic version of the ASP, refer to e.g. [1] and [15] for a detailed description regarding appropriate modifications of formulated MIP.

2.4 The Aircraft Sequencing Problem as Routing Problem

In principle, the ASP can be viewed as a routing problem. This representation, although already chosen in former literature (under a different objective), is not indisputable.[4]

The single runway case of the ASP is similar to a Travelling Salesman Problem with Time Windows (TSPTW). The multiple runway case corresponds to a Vehicle Routing Problem with Time Windows (VRPTW), with the different vehicle routes being the different runways.

[4] See e.g. [1] p. 188.

Each vertex on the graph defining a routing problem corresponds to an airplane with associated time window. Each arc represents the required minimum successive separation between respective nodes, i.e. consecutive airplanes. Note that the vertex that represents the depot does not have any meaning in the sense of an ASP.

If the minimum separation matrix of involved airplanes, i.e. the distance matrix, is symmetrical, one faces a symmetrical routing problem. Otherwise, it is defined as being asymmetrical. Further note, as long as the separation matrix fulfils the triangle inequality, successive separation is sufficient to guarantee complete separation, and the formulation as routing problem is straightforward. Otherwise, a formal representation is getting more complicated and requires additional considerations to ensure complete separation criteria.

With respect to the objective function adopted, the ASP is a fairly unusual application of a routing problem. In fact, it can be interpreted as a routing problem with earliness and tardiness penalties for customers not being visited at individually specified points in time, rather than within individually specified periods of time.

2.5 Problem Complexity

As the formulation of the ASP as routing problem is possible, it is straightforward to determine its complexity.

With reference to [5], the single runway ALP minimizing total delay is equivalent to a Cumulative Travelling Salesman Problem with Ready Times (CTSP-RT). As already the general case of the TSP is known to be NP-hard, the single runway ALP (independent of objective function) and all extensions to it must be at least NP-hard as well. In fact, in [5] the author further proved the affiliation of the ALP to the class of strongly NP-hard optimization problems by the formulation as Job-Shop Scheduling Problem (JSS).

In other words, no solution method exists able to solve the ASP to optimality in polynomial time.

3. Variable Neighborhood Search for the Static Aircraft Sequencing Problem

As the ASP belongs to the class of strongly NP-hard optimization problems, heuristics are commonly applied as solution methods. In this paper, Variable Neighborhood Search (VNS) as proposed by [14] is adopted.

3.1 Basic Variable Neighborhood Search

VNS is a recent metaheuristic successfully proved to solve combinatorial optimization problems. Its basic idea is to randomly or systematically explore a set of pre-defined neighborhoods, typically arranged in one another, both in finding local optima as well as escaping from them. VNS simply uses the fact that a global optimum corresponds to a local optimum for a specific neighborhood. In using different neighborhoods, different landscapes in the solution space will be generated, enhancing the possibility of finding the optimal solution.

In literature, several variants of VNS exist. In this work, the Basic VNS scheme as proposed by [14] is adopted (see Algorithm 3.1). Its main characteristic is the combination of stochastic and systematic elements. Starting from any given initial solution, a shaking step is performed generating a random solution from the first neighborhood. It follows a local search algorithm, aiming to obtain a local optimum. If this candidate solution is better than the incumbent (=current) solution, the algorithm starts again with the first neighborhood, which is now applied to the new current solution. Otherwise, or if no new current solution was found, the algorithm switches to the next larger neighborhood, and again performs a shaking step, followed by local search. This procedure is repeated until the stopping condition, being a limit on the number of iterations, a CPU time limit, or a limit on the number of iterations between two improvements, is met. In finding a proper balance between diversification and intensification of the search, accomplished by shaking and local search respectively, the number and size of the neighborhoods are important design decisions. For a more detailed explanation on VNS refer to e.g. [10] and [14].

Basic Variable Neighborhood Search

Initialization: Select the set of neighborhood structures N_k, for $k = 1, \dots, k_{max}$, that will be used in the search; find an initial solution x; choose a stopping condition;

Repeat the following sequence until the stopping condition is met:

(1) Set $k \leftarrow 1$;

(2) *Repeat* the following steps until $k = k_{max}$:

(a) Shaking. Generate a point x' at random from the k^{th} neighborhood of x ($x' \in N_k(x)$);

(b) Local search. Apply some local search method with x' as initial solution; denote with x'' the so obtained local optimum;

(c) Move or not. If the local optimum x'' is better than the incumbent x, move there ($x \leftarrow x''$), and continue the search with $N_1(k \leftarrow 1)$; otherwise, set $k \leftarrow k + 1$;

Algorithm 3.1: Steps of the Basic Variable Neighborhood Search.[5]

3.2 Implementation Details

As the ASP in thesis will be considered as routing problem, design decisions of the implemented algorithm are mainly related to works of [11], [16], [17] and [18]. Nevertheless, problem-specific knowledge mainly related to [19] was incorporated.

3.2.1 Pseudo Code of the Implemented Variable Neighborhood Search

Algorithm 3.2 presents the pseudo code of the implemented VNS. In fact, its main steps correspond to the VNS scheme explained above. In principal, the pseudo code is self-explanatory. Nevertheless, each procedural step and related design decisions are described in full detail on the following pages.

[5] Cf. [10] p. 10.

Implemented Variable Neighborhood Search

$x \leftarrow$ `buildInitialSolution();` // generate the initial solution
$x_{best} \leftarrow x$; $k \leftarrow 1$; $iteration \leftarrow 0$; $lapTime \leftarrow start\ timer$ // initialization
while ($iteration < VNS\_ITERATIONS$) and ($lapTime < RUNTIME\_LIMIT$) do:
    if ($NUMBER\_OF\_RUNWAYS > 1$) do:
        $runway1, runway2 \leftarrow$ `get2Runways();` // randomly select 2 runways
    $x' \leftarrow$ `doShaking(`$x, runway1, runway2, k$`);` // perform shaking
    $x'' \leftarrow$ `do2Opt(`$x', runway1, runway2$`);` // perform local search
    if ($f(x'') \leq (1 + \theta) \cdot f(x)$) do: // acceptance decision
        $x \leftarrow x''$; $k \leftarrow 1$;
        if ($f(x'') \leq f(x_{best})$) do:
            $x_{best} \leftarrow x''$;
    else $k \leftarrow 1 + (k \,\%\, k_{max})$;
    $iteration++$;
return x_{best}; // output

Algorithm 3.2: Pseudo code of the implemented Variable Neighborhood Search.

3.2.2 Data Structures and Solution Representation

The choice of suitable data structures and an appropriate representation of solutions are vital design decisions when developing an algorithm.

3.2.2.1 Airplane-Specific Input Data

Figure 3.1 presents the data structure incorporated to hold all input data of a set of airplanes. For each airplane, five variables, being the earliest, target and latest times as well as early and late marginal penalty costs, are stored.

Airplanes i	0	1	2	3	4	5	6	7	8	9
E_i	..	..	..	..	..	..	..	..	..	..
T_i	..	..	..	..	..	..	..	..	..	..
L_i	..	..	..	..	..	..	..	..	..	..
B_i	..	..	..	..	..	..	..	..	..	..
A_i	..	..	..	..	..	..	..	..	..	..

Figure 3.1: Data Structure to store input data of a set of airplanes.

3.2.2.2 Minimum Separation Matrix

The design of the data structure holding the minimum separation matrix is presented in Figure 3.2. Required successive separations between predecessors and successors are stored as times. Note that whenever the predecessor equals the successor, a very large value is associated with minimum separation time.

Predecessors \ Successors	0	1	2	3	4	5
0	∞	..	..	..	..	..
1	..	∞	..	..	..	..
2	..	..	∞	..	..	..
3	..	..	..	∞	..	..
4	..	..	..	..	∞	..
5	..	..	..	..	..	∞

Figure 3.2: Structure of the minimum separation matrix.

3.2.2.3 Solution Representation

Any solution must contain information regarding each runway´s sequence and each airplane´s scheduled time.

For each runway, the sequence is simply represented by the order in which airplanes are computed to land or depart. Figure 3.3 shows an example of the representation of two runway sequences.

Runway 0	0	3	4	6	7

Runway 1	2	1	5	8	9

Figure 3.3: Representation of two runway sequences.

In addition, information of each airplane´s allocated landing or take-off time, i.e. the solution schedule, is stored in a data structure (see Figure 3.4).

Airplanes i	0	1	2	3	4	5	6	7	8	9
t_i	..	..	..	..	..	..	..	..	..	..

Figure 3.4: Representation of a solution schedule.

In other words, any solution obtained in the course of computation consists of two parts: the totality of the stored runway sequences and the schedule of all airplanes involved.

3.2.3 Scheduling

Each scheduling process in the algorithm is performed by solving the MIP introduced. Given the sequences, an optimal schedule of involved airplanes can be computed. Note that if the sequences are determined, all binary variables are known, and the MIP transforms into a simple LP. This in turn can be solved to optimality in polynomial time.

A "Mixed Integer Linear Programming (MILP)"-solver[6] was adopted to compute the LPs. Nevertheless, it is still possible that the LP for any sequence is infeasible, i.e. either time window or complete separation criteria is violated. Expressed in mathematical terms, not both of the following conditions hold for all airplanes:

Time window constraints:	$E_i \leq t_i \leq L_i$	$\forall i \in P$	(24)
Separation constraints:	$t_i + S_{ij} \leq t_j$	$\forall i, j \in P, t_i \leq t_j$	(25)

Note that the implemented solver is unable to handle infeasibilities. As VNS is able to handle them, infeasible solutions are accepted anyway. Nevertheless, they have to be penalized. Therefore, if a LP cannot be solved, the scheduling algorithm assigns to each airplane being part of such a sequence a large positive constant value as its individual scheduled time. This ensures that the infeasibility will be considered when calculating costs. The particular handling of invalid solutions is topic of a later chapter.

3.2.4 Initial Solution

In obtaining an initial solution, a very simple and intuitive procedure, summarized in Algorithm 3.3, was adopted.

Initially, all airplanes will be pre-ordered according to non-decreasing target times. In the single runway case, the resulting sequence already provides the initial solution. In the presence of multiple runways, all pre-ordered airplanes are assigned to available runways following a round robin procedure: The first airplane in the pre-ordered sequence will be

[6] See [13].

allocated to the first runway, the second airplane to the second runway, and so on. These steps are repeated, until no empty runways are available anymore, i.e. exactly one airplane is allocated to each runway. The algorithm then switches back to the first runway and continues to insert the next airplane at the end of the existing sequence. This whole procedure is repeatedly applied, until all airplanes are assigned to exactly one runway. Figure 3.5 provides a graphical illustration of the described assignment algorithm.

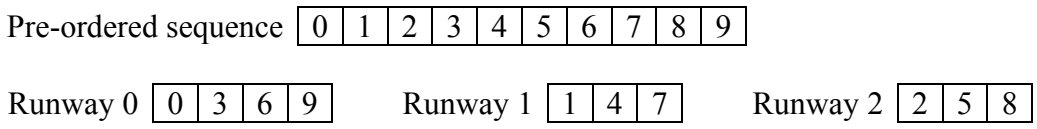

Figure 3.5: Round robin runway assignment.

The last step in generating an initial solution is the computation of a schedule. To provide a feasible initial solution, each LP of involved sequences must be feasible. Anyway, as the proposed VNS is able to handle infeasibilities, not much effort is taken in trying to come up with a feasible initial solution. Moreover, the main concern lies in keeping the initial solution algorithm as fast and simple as possible.

Generation of the initial solution: `buildInitialSolution()`

$s_{preordered}$ ← pre-order all airplanes $i \in P$ according non-decreasing target times;
if ($NUMBER\_OF\_RUNWAYS = 1$) do: // single runway case
    return $x \leftarrow s_{preordered}$; // pre-ordered sequence equals runway sequence
else do: // multiple runway case
    select first airplane $i \in s_{preordered}$;
    while not all airplanes $i \in s_{preordered}$ are assigned to a runway $r \in R$ do:
        $r \leftarrow 0$; // select first runway
        while ($r < m$) do: // add exactly one airplane to each runway
            insert selected airplane $i \in s_{preordered}$ at the end of sequence s_r;
            select next runway $r \leftarrow r + 1$;
            select next airplane $i \in s_{preordered}$;
    return $x \leftarrow s_r, \forall r \in R$; // solution represents set of runway sequences

Algorithm 3.3: Generation of the initial solution.

3.2.5 Shaking

3.2.5.1 Shaking Operators

The selection of an appropriate set of neighborhood structures is the core part in designing a VNS. A neighborhood of the incumbent solution is defined by means of a function or an operator. On the one hand the aim of the operator is to sufficiently perturb the current solution on the other hand it should keep the good parts of it.

The implemented VNS applies the two well-known "move" and "cross-exchange" operators. They are very common when dealing with routing problems because they usually provide very effective neighborhoods. Figures 3.6 and 3.7 graphically demonstrate their operations on the basis of move and cross-exchanges between two sequences: The move operator removes the segment $(x_2{}',y_2)$ of one sequence and inserts it into another sequence, while the cross-exchange operator swaps two segments $(x_1{}',y_1)$ and $(x_2{}',y_2)$ of different sequences. Thereby the orientation of segments and sequences are preserved.

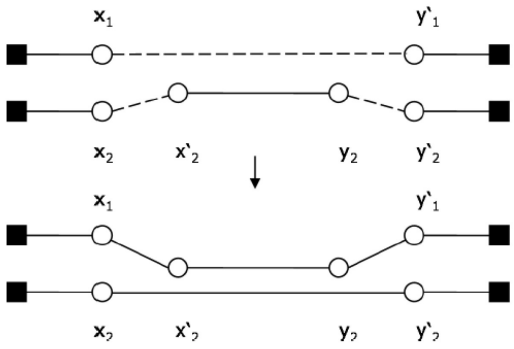

Figure 3.6: The move operator.[7]

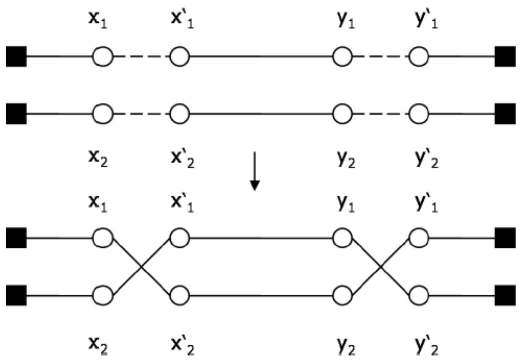

Figure 3.7: The cross-exchange operator.[8]

[7] Cf. [11] p. 793.

[8] Cf. [11] p. 793.

3.2.5.2 Shaking Neighborhood Order

In designing an appropriate shaking neighborhood order, problem-specific knowledge was incorporated. In other words, the level of diversification, i.e. the distance from the incumbent solution, of each shaking step is controlled by both the maximum number of positions airplanes can be actively shifted, and the maximum number of airplanes that are allowed to be relocated. To stepwise increase the level of disturbance, 18 different neighborhoods, arranged in a fixed order, were adopted (see Table 3.1).

k	*Operator*	*Minimum active position shift*	*Maximum active position shift*	*Minimum segment length*	*Maximum segment length*[9]
1	move	0 or 1	1	1	min(1, u)
2	move	0 or 1	1	1	min(2, u)
3	move	0 or 1	1	1	min(3, u)
4	move	0 or 1	2	1	min(1, u)
5	move	0 or 1	2	1	min(2, u)
6	move	0 or 1	2	1	min(3, u)
7	move	0 or 1	3	1	min(1, u)
8	move	0 or 1	3	1	min(2, u)
9	move	0 or 1	3, 5 or unrestricted	1	min(3 or 5, u)
10	cross	0 or 1	1	1	min(1, u)
11	cross	0 or 1	1	1	min(2, u)
12	cross	0 or 1	1	1	min(3, u)
13	cross	0 or 1	2	1	min(1, u)
14	cross	0 or 1	2	1	min(2, u)
15	cross	0 or 1	2	1	min(3, u)
16	cross	0 or 1	3	1	min(1, u)
17	cross	0 or 1	3	1	min(2, u)
18	cross	0 or 1	3, 5 or unrestricted	1	min(3 or 5, u)

Table 3.1: Shaking neighborhood order.

In principle, for values of $k \leq 9$ the move operator is applied, while for values of $k > 9$ the cross-exchange operator determines the neighborhood.

The minimum number of positions an airplane has to be actively shifted during each shaking step is “1” for the single runway case and “0” for the multiple runway case. The maximum number of positions an airplane is allowed to be actively shifted depends on k and is bounded by the values depicted in the respective column. For values of $k = 9$ and

[9] u... number of elements until end of sequence, beginning of next segment to be relocated, or insertion point.

$k = 18$, the maximum values depend on the parameter setup (“3”, “5” or “unrestricted”). This enables a possible higher level of diversification.

The number of positions a segment is actually shifted relates to its current position in the sequence and is randomly determined. In other words, if a single airplane is located at position “7” and is actively moved to another runway with a maximum active position shift of “2”, it can be inserted at positions “5”, “6”, “7”, “8” or “9”. If the size of the other sequence is e.g. “3”, it will be inserted at the end.

In this context, the term “active” refers to the active repositioning of a segment of airplanes. In contrast, airplanes not involved in a move or swap may be driven out of their positions “passively” (by others being actively relocated) by even more than the maximum allowed value of active position shift.

The minimum length of a segment to be moved or swapped is always “1”. The maximum segment length depends on k and is shown in the respective column. For values of $k = 9$ and $k = 18$, the maximum segment length again varies in accordance with applied parameter setup (“3” or “5”) to allow a possible higher level of diversification.

The actual segment length is determined randomly in the interval $[1, \min(x, u)]$, where x represent the value as depicted in the respective column. The term “u” refers either to the number of elements remaining until the end of the respective sequence, the beginning of the next segment to be relocated, or the position of the insertion point of the segment. Note that the two latter in fact have on a procedural background: As starting and insertion points of segments to be relocated are always selected first by the shaking algorithm, these additional bounds already exist when determining the segment lengths.

Finally note that the shaking algorithm does not check whether a shift may violate any time window constraint.

3.2.5.3 Pseudo Code of the Shaking Algorithm

Algorithm 3.4 presents the pseudo-code of the shaking algorithm. In principle, in the single runway case, moves and cross-exchanges only occur within the sequence considered. In the presence of multiple runways, they will only be performed between two randomly selected sequences.

For the sake of thorough understanding, each case of the implemented shaking algorithm is explained in full detail on the following pages.

Shaking algorithm: `doShaking(`$x, runway1, runway2, k$`)`

if ($NUMBER\_OF\_RUNWAYS = 1$) do: // single runway case
    if ($k \leq k_{max}/2$) do: // move a random segment within the sequence;
        $x' \leftarrow$ `doIntraSequenceMove(`x, k`);`
    if ($k > k_{max}/2$) do: // cross-exchange two random segments within the sequence;
        $x' \leftarrow$ `doIntraSequenceSwap(`x, k`);`
if ($NUMBER\_OF\_RUNWAYS \geq 2$) do: // multiple runway case
    if ($k \leq k_{max}/2$) do: // move a random segment between two runways;
        $x' \leftarrow$ `doInterSequenceMove(`$x, runway1, runway2, k$`);`
    if ($k > k_{max}/2$) do: // cross-exchange two random segments between two runways;
        $x' \leftarrow$ `doInterSequenceSwap(`$x, runway1, runway2, k$`);`
return x'; // disturbed solution

Algorithm 3.4: Pseudo code of the shaking algorithm.

3.2.5.4 Move in the Single Runway Case

In the single runway case, a move shifts a randomly selected segment of airplanes to a random position within the sequence, while observing lower and upper bounds for active position shift and segment length. Figure 3.8 illustrates an intra-sequential move.

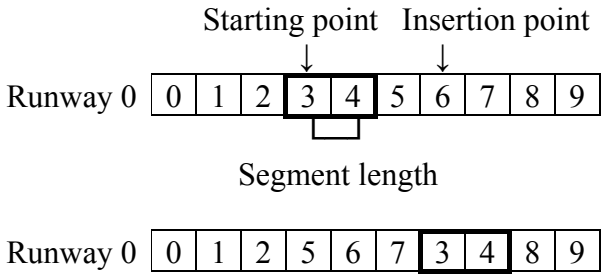

Figure 3.8: Move within a sequence.

As starting and insertion points are always determined first, in the example above, the segment length may be additionally constrained by the position of the insertion point.

3.2.5.5 Cross-Exchange in the Single Runway Case

In the single runway case, a cross-exchange swaps two randomly selected segments of airplanes within the sequence, while observing lower and upper bounds for active position shift and segment lengths. Figure 3.9 provides an example of an intra-sequential swap.

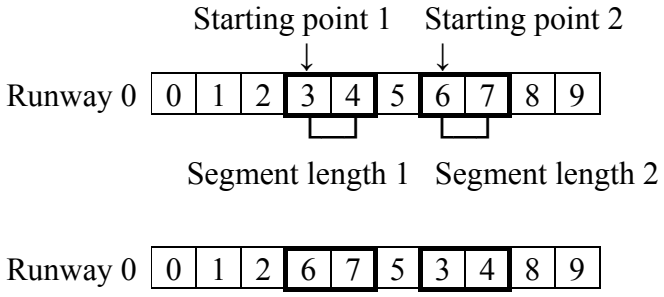

Figure 3.9: Cross-exchange within a sequence.

As the starting points are determined first, in the example above, segment length "1" may be additionally bounded by the position of starting point "2", while segment length "2" may also be constrained by the end of the sequence. To inhibit excessive disturbances, the algorithm uses identical lengths of the segments involved, i.e. the lower segment size dominates.

3.2.5.6 Move in the Multiple Runway Case

In the multiple runway case, a move shifts a randomly determined segment of airplanes from one randomly selected sequence to a random position in another randomly selected sequence, while observing lower and upper bounds for active position shift and segment length. Figure 3.10 illustrates an inter-sequential move.

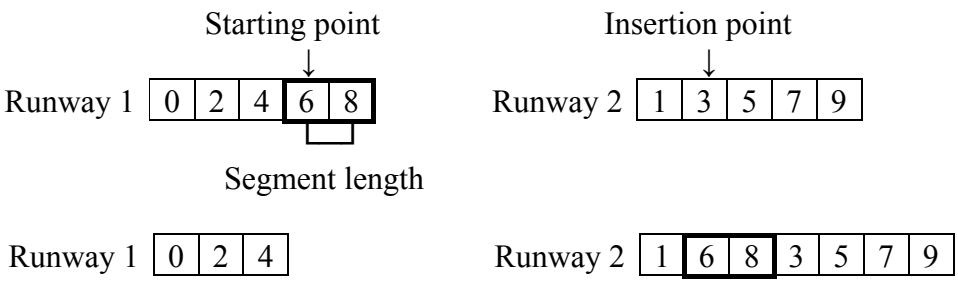

Figure 3.10: Move between two sequences.

In a pre-processing step the two runways involved are randomly selected. By convention, a move always takes place from the first selected sequence to the second one. Hence, the starting point and length of the segment to be moved are always randomly determined on the first selected sequence. In the example above, the segment length may be additionally bounded by the end of sequence “1”.

3.2.5.7 Cross-Exchange in the Multiple Runway Case

In the presence of multiple runways, a cross-exchange swaps two randomly determined segments of airplanes between two randomly selected sequences, while observing lower and upper bounds for active position shift and segment lengths. Figure 3.11 provides an example of an inter-sequential swap.

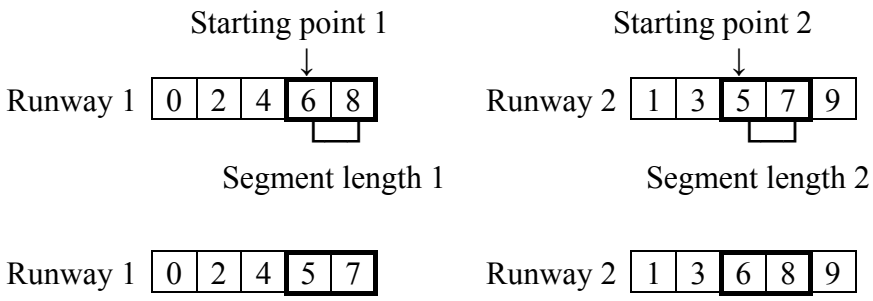

Figure 3.11: Cross-exchange between two sequences.

In a pre-processing step the two runways involved are randomly selected. In the example above, the segment lengths may be additionally constrained by the end of the respective sequences. To prevent excessive disturbances, the algorithm uses identical segment lengths, i.e. the smaller segment size dominates.

3.2.6 Local Search

Local search is applied to each sequence disturbed during the shaking phase. Its aim is to find a local optimum in iteratively improving the solution obtained through shaking.

3.2.6.1 2-opt Operator

A simple 2-opt first improvement strategy is adopted, while visiting the neighborhoods in a lexicographic order. In other words, the local search algorithm restarts instantly if an improving move was found. The lexicographic search strategy guarantees a systematic, hence efficient, visit of the neighborhood. With the 2-opt operator, every single move in fact corresponds to an inversion of a segment of the respective sequence (see Figure 3.12). To limit runtime, a bound on the segment length to be inversed was defined. In detail, the segment length parameter λ was chosen to be "2" or "5", depending on adopted parameter setup. This reduces the size of the neighborhood significantly.

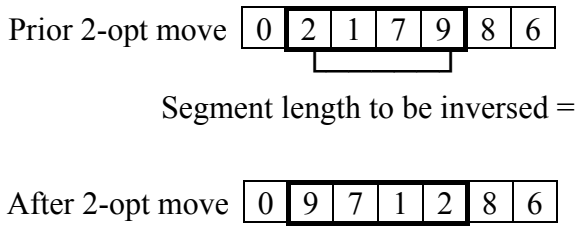

Figure 3.12: 2-opt move.

3.2.6.2 Pseudo Code of the Local Search Algorithm

Algorithm 3.5 provides the pseudo code of the implemented 2-opt procedure. Note that the algorithm is applied only to those sequences disturbed during the shaking phase.

To enable the evaluation of a 2-opt move, each step requires solving a LP. Especially for large instances this can be a very time-consuming process. Furthermore, a considerable amount of schedules may have to be computed during the course of local search. Hence, to limit computation time, strategies have to be adopted able to recognize promising neighbours in advance. In detail, these are pre-infeasibility and pre-profitability checks.[10]

[10] See [19] p. 78-82.

Only if the pre-infeasibility check fails and the pre-profitability check assumes a gain from the approaching 2-opt step, the move is actually executed.

A performed 2-opt move is only accepted, if the total cost of the altered sequence is actually lower than the cost of the current sequence, i.e. if:

$$c(s_r') < c(s_r). \tag{26}$$

Recall, if any LP of involved sequences fails to be solved, i.e. the respective sequence is infeasible, all airplanes belonging to that sequence are assigned large positive constant integer values representing their scheduled times. In this way, any infeasible sequence is identified by their respective cost function. Nevertheless, penalization of infeasible sequences in the course of local search is not performed.

2-opt Local Search: `do2Opt(`$x', runway1, runway2$`)`

for ($r = runway1, runway2$) do: // runways disturbed by shaking
    $s_r \leftarrow x'[r]$; // sequence of respective runway
    repeat the following until no better sequence is found:
        for ($i \leftarrow 0$ to $|s_r| - 2$) do: // outer loop: 1^{st} element to element before last
            for ($j \leftarrow i + 2$ to $|s_r|$) do: // inner loop: 2^{nd} to element after last
                if ($|j - i| > \lambda$) exit j-loop; // to limit runtime
                perform pre-infeasibility check of 2-opt move;
                perform pre-profitability check of 2-opt move;
                if move is not infeasible and profitability is estimated do:
                    s_r' ← inverse segment from i to $j - 1$;
                    if ($c(s_r') < c(s_r)$) do: // acceptance decision
                        $s_r \leftarrow s_r'$;
                        goto outer loop and continue with $i \leftarrow 0$;
                    else s_r ← re-inverse segment from i to $j - 1$;
    $x''[r] \leftarrow s_r$; // insert 2-optimal sequence in solution
return x''; // optimized solution

Algorithm 3.5: 2-opt local search algorithm.

3.2.6.3 Pre-Infeasibility Check[11]

The aim of the pre-infeasibility check is to identify those neighbors that are necessarily infeasible with respect to their time windows.

In other words, when considering two airplanes i and j, being the only ones involved in a 2-opt move, with i being originally sequenced immediately ahead of j, the move, i.e. the inversion, is necessarily infeasible, if: $L_i < E_j$. The same applies in the general sense for any segment of airplanes being inversed. Figure 3.13 provides a graphical illustration.

Sequence prior 2-opt move

i	0	2	1	7	9	8	6
E_i	E_0	E_2	E_1	E_7	E_9	E_8	E_6
L_i	L_0	L_2	L_1	L_7	L_9	L_8	L_6

Sequence after 2-opt move

i	0	9	7	1	2	8	6
E_i	E_0	E_9	E_7	E_1	E_2	E_8	E_6
L_i	L_0	L_9	L_7	L_1	L_2	L_8	L_6

Figure 3.13: Illustration of pre-infeasibility check.

In the example above, a 2-opt move would be necessarily infeasible, if any of the following conditions apply.

$$L_2 < E_1, L_2 < E_7, L_2 < E_9, L_1 < E_7, L_1 < E_9 \text{ or } L_7 < E_9. \qquad (27)$$

3.2.6.4 Pre-Profitability Check[12]

The pre-profitability check corresponds to an estimation of the gain or loss of a 2-opt move.

For the sake of simplicity, it will be assumed that, when considering two airplanes i and j scheduled at t_i and t_j respectively, being the only ones involved in a 2-opt move, airplane i

[11] See [19] pp. 78-82.

[12] See [19] p. 82.

is scheduled at t_j and airplane j is schedule at t_i after being inversed. A profitable move in this sense is expected if the following applies:

$$f_i(t_i) + f_j(t_j) - [f_i(t_j) + f_j(t_i)] > 0 \qquad (28)$$

Of course, the same is true when considering any segment of airplanes to be inversed. Figure 3.14 illustrates an example of a pre-profitability check.

Sequence prior 2-opt move

i	0	2	1	7	9	8	6
t_i	t_0	t_2	t_1	t_7	t_9	t_8	t_6

Sequence after 2-opt move

i	0	9	7	1	2	8	6
t_i	t_0	t_2	t_1	t_7	t_9	t_8	t_6

Figure 3.14: Illustration of pre-profitability check.

In the example above, a gain is estimated, and the move is accepted, only if:

$$f_2(t_2) + f_1(t_1) + f_7(t_7) + f_9(t_9) - [f_9(t_2) + f_7(t_1) + f_1(t_7) + f_2(t_9)] > 0. \qquad (29)$$

Note that this procedure only represents a rough estimation. It is neither assured that an accepted move is actually profitable, nor that any move rejected might not have been advantageous.

3.2.7 Acceptance Decision

After the shaking and local search phase, the candidate solution x'' obtained has to be compared to the incumbent solution x to decide whether to accept it or not.

The candidate solution x'' is accepted, if its associated evaluation value $f(x'')$ does not exceed $(100 + \theta)\%$ of the incumbent solution´s evaluation value $f(x)$. Expressed in mathematical terms, the candidate solution is accepted as new incumbent solution only if:

$$f(x'') \leq (1+\theta) \cdot f(x) \qquad \text{with } 0 \leq \theta \leq 1. \qquad (30)$$

As infeasible solutions are allowed in principal, techniques to drive the search into feasible regions must be incorporated. In fact, this is realized by simply penalizing invalid solutions. Recall that the scheduling algorithm assigns each airplane of any infeasible sequence a large positive constant value, being its individual scheduled time. This in turn will not only be interpreted by the cost function $c(x'')$, which in fact corresponds to the objective function introduced, but also by an additionally incorporated penalty function $w(x'')$ as all airplanes assigned to that specific runway having violated their respective time windows. The total time window violation, expressed as the penalty value $w(x'')$, will further be multiplied by the penalty parameter $\alpha \geq 0$. The resulting sum of cost function $c(x'')$ and weighted penalty function $\alpha \cdot w(x'')$ represents the evaluation function $f(x'')$ of the candidate solution x''. Formulas (31) - (33) provide a mathematical summary.

Cost function: $$c(x'') = \sum_{i=1}^{n} (B_i \cdot b_i + A_i \cdot a_i) \qquad (31)$$

Penalty function: $$w(x'') = \sum_{i=1}^{n} \max(0, t_i - L_i) \qquad (32)$$

Evaluation function: $$f(x'') = c(x'') + \alpha \cdot w(x'') \qquad (33)$$

In fact, this procedure allows the identification of infeasible solutions. In this way, it enables the algorithm to drive the search into feasible regions of the solution space.

3.2.8 Stopping Condition

In principle, the number of iterations was chosen as stopping condition. Nevertheless, to prevent excessive runtimes, a computation time limit was implemented additionally.

In other words, the algorithm stops if a given number of iterations were performed or the runtime limit of 7200 seconds (2 hours) is exceeded, whichever occurs earlier.

4. Computational Experiments

The algorithm developed was programmed in C++ and run on a Home-PC with Intel® Core™ i5 CPU 760@2.8GHz processor, 8GB of memory and 64 bit operating system.

The LPs in the course of the implemented VNS were solved using "lp_solve 5.5"-software, a free "Mixed Integer Linear Programming (MILP)"-solver available from [13].

In this section, computational results for 13 publicly available test sets from J.E. Beasley´s OR-library, available from [4], involving from 10 to 500 airplanes with a varying number of runways, are reported. This gives 49 instances in total.

Further, computational results of the application of the implemented VNS to a real world decision problem, encountered at Munich Airport, are presented.

4.1 Computational Parameters

For the computational experiments, six different parameter settings, reaching from a rather restrictive to a relatively "laissez-faire" setup, were adopted (see Table 4.1).

Setup	$\boldsymbol{\lambda_{2opt}}$	*Maximum active position shift*	*Maximum segment length*
1	2	3	3
2	2	5	5
3	2	unrestricted	5
4	5	3	3
5	5	5	5
6	5	unrestricted	5

Table 4.1: Parameter settings.

Additionally, the following parameters, related to acceptance decision and penalizing function respectively, were chosen: $\theta = 0.05$ (5%) and $\alpha = 10000$.

To reduce computational time, the number of iterations was limited to 100 and 1000 respectively. Preliminary tests revealed that these are sufficient to produce stable high quality solutions.

The heuristic was run ten times per problem instance. Average values rounded to hundredths are reported. Total computation times were rounded to integer values.

4.2 Test Problems

To measure the performance of the underlying metaheuristic, test problems publicly available from J.E. Beasley´s OR-library, available from [4], were used. These often called "Beasley"-test instances serve as benchmark in most papers dealing with the ASP.

4.2.1 Characteristics of Input Data

The test sets consist of eight small sets, involving from 10 to 50 airplanes, and five large sets, involving from 100 to 500 airplanes, each with a varying number of runways. In total, this gives 25 small and 24 large instances.

In fact, the underlying data files only contain arriving aircraft, i.e. they cover the ALP only. Their separation matrices are both symmetrical and asymmetrical, with and without fulfillment of the triangle inequality. Table 4.2 summarizes the main characteristics of the input data files. For more details regarding the structure of the files refer to [4].

Set	*n*	*Separation matrix*	*Triangle inequality*
1	10	symmetrical	fulfilled
2	15	symmetrical	fulfilled
3	20	symmetrical	fulfilled
4	20	symmetrical	fulfilled
5	20	symmetrical	fulfilled
6	30	asymmetrical	fulfilled
7	44	asymmetrical	fulfilled
8	50	symmetrical	not fulfilled
9	100	asymmetrical	fulfilled
10	150	asymmetrical	fulfilled
11	200	asymmetrical	fulfilled
12	250	asymmetrical	fulfilled
13	500	asymmetrical	fulfilled

Table 4.2: Characteristics of test sets.

4.2.2 Characteristics of Benchmark Results

Optimal results for the small instances of the test problems are given in [1] and [15]. For the large instances optimal results does not exist. The best (-known) results found by the "the test sets publishing"-authors in the course of their computational work are presented in [15]. These results are commonly used in literature as benchmark. Although recently some new best-known results were obtained by [20], for reasons of consistency and easier comparability, and because of the fact that [20] is still unpublished[13], the results found by [15] also serve as benchmark in the underlying work. Nevertheless, a comparative study is done in a later chapter, acknowledging recently found new best solutions.

In computing the benchmark results, each test problem was solved with an increasing number of runways until the solution value dropped to zero. That indicates sufficient runways are available to enable all airplanes to land at target time. Also note that for the multiple runway cases independent runway operations were assumed, i.e. $s_{ij} = 0$. To enable comparability, the same procedure had to be adopted in this computational study.

4.2.3 Results of the Small Instances

4.2.3.1 Detailed Results and Interpretation

Aggregated results for all six parameter settings are shown in Table 4.5 in the next subchapter. As parameter setting "1" provided the least computation times paired with optimal solutions, results only for this setup are presented in detail; each for 100 and 1000 iterations (see Tables 4.3 and 4.4 respectively).

All computed solutions of the small test instances were feasible. As can be recognized, the initial solution already represents the optimal solution in 15 of the 25 small instances. This indicates that the algorithm for generating an initial solution in both the single and the multiple runway cases produce good starting solutions for the improvement heuristic. Its computation time in general lies within a fraction of a second. The largest computation time was 0.06 seconds for problem set "8" involving a single runway. Note that the required CPU-time to find an initial solution is mainly influenced by the size and characteristics of the LP that has to be solved.

[13] Note that results were only presented on "Chinese Control and Decision Conference 2011".

VNS was able to find optimal solutions for all small test instances. It shows that it is capable of producing very stable results, i.e. Z_{avg} equals Z_{min}, for almost every instance. The largest average computation times of the improvement heuristic were generated for problem set “8” involving a single runway, with 12.77 seconds for 100 iterations and 129.17 seconds for 1000 iterations. This corresponds to average computation times per airplane of 0.26 seconds and 2.58 seconds respectively.

Set	n	m	Z_{opt}	Z_{init}	Z_{min}	Z_{max}	Z_{avg}	t_{avg}	T	RPD_{min}
1	10	1	700	700	700	700	700	1.50	15	0
		2	90	90	90	90	90	1.41	14	0
		3	0	0	0	0	0	1.07	11	0
2	15	1	1480	1500	1480	1480	1480	2.12	21	0
		2	210	210	210	210	210	1.75	18	0
		3	0	0	0	0	0	1.34	14	0
3	20	1	820	1730	820	820	820	2.36	24	0
		2	60	120	60	60	60	2.44	24	0
		3	0	0	0	0	0	1.54	15	0
4	20	1	2520	2520	2520	2520	2520	2.52	25	0
		2	640	640	640	640	640	1.82	18	0
		3	130	130	130	130	130	1.53	15	0
		4	0	0	0	0	0	1.38	14	0
5	20	1	3100	5420	3100	3100	3100	2.55	26	0
		2	650	1140	650	670	652	2.38	24	0
		3	170	270	170	170	170	1.57	16	0
		4	0	0	0	0	0	1.34	13	0
6	30	1	24442	24442	24442	24442	24442	2.68	27	0
		2	554	1011	554	697	568.3	2.39	24	0
		3	0	0	0	0	0	1.22	12	0
7	44	1	1550	1550	1550	1550	1550	6.54	65	0
		2	0	0	0	0	0	2.96	30	0
8	50	1	1950	2480	1950	1950	1950	12.77	128	0
		2	135	285	135	135	135	6.51	65	0
		3	0	15	0	0	0	2.71	27	0
							Mean Value	2.74	27	0

Table 4.3: Results of small instances with parameter setup "1" and 100 iterations.

Set	*n*	*m*	Z_{opt}	Z_{init}	Z_{min}	Z_{max}	Z_{avg}	t_{avg}	T	RPD_{min}
1	10	1	700	700	700	700	700	15.44	154	0
		2	90	90	90	90	90	14.16	142	0
		3	0	0	0	0	0	10.56	106	0
2	15	1	1480	1500	1480	1480	1480	21.50	215	0
		2	210	210	210	210	210	17.16	172	0
		3	0	0	0	0	0	13.42	134	0
3	20	1	820	1730	820	820	820	23.55	236	0
		2	60	120	60	60	60	24.48	245	0
		3	0	0	0	0	0	15.17	152	0
4	20	1	2520	2520	2520	2520	2520	26.02	260	0
		2	640	640	640	640	640	18.05	181	0
		3	130	130	130	130	130	15.29	153	0
		4	0	0	0	0	0	13.49	135	0
5	20	1	3100	5420	3100	3100	3100	25.50	255	0
		2	650	1140	650	650	650	24.33	243	0
		3	170	270	170	240	198	15.61	156	0
		4	0	0	0	0	0	13.45	135	0
6	30	1	24442	24442	24442	24442	24442	26.74	267	0
		2	554	1011	554	554	554	22.31	223	0
		3	0	0	0	0	0	12.38	124	0
7	44	1	1550	1550	1550	1550	1550	65.84	658	0
		2	0	0	0	0	0	29.64	296	0
8	50	1	1950	2480	1950	1950	1950	129.17	1292	0
		2	135	285	135	135	135	64.54	645	0
		3	0	15	0	0	0	27.02	270	0
							Mean Value	27.39	274	0

Table 4.4: Results of small instances with parameter setup "1" and 1000 iterations.

4.2.3.2 Parameter-Specific Aggregated Results

Table 4.5 summarizes aggregated results of all small instances in dependence of parameter setup and number of iterations.

It shows that runtimes $t_{avg_{mean}}$ and T_{mean} are constantly increasing by stepwise unleashing the search, while values of $RPD_{min_{mean}}$ remain optimal. Hence, the best tradeoff when dealing with small instances seems to be the choice of parameter setup "1" with 100 iterations. Note that, if the number of iterations is increased by a factor of "10", mean vales of computation times will also be higher by a factor of approximately "10".

Setup	*VNS 100 iterations*			*VNS 1000 iterations*		
	$t_{avg_{mean}}$	T_{mean}	$RPD_{min_{mean}}$	$t_{avg_{mean}}$	T_{mean}	$RPD_{min_{mean}}$
1	2.74	27	0	27.39	274	0
2	2.92	29	0	28.44	284	0
3	3.48	35	0	35.38	354	0
4	3.41	34	0	34.84	348	0
5	3.66	37	0	36.13	361	0
6	4.52	45	0	45.57	456	0

Table 4.5: Aggregated results of small instances.

4.2.4 Results of the Large Instances

4.2.4.1 Detailed Results and Interpretation

In the next subchapter, Table 4.8 summarizes the aggregated results of the large instances for adopted parameter settings "1" and "5". As setup "1" provided the least aggregated RPD_{min}, i.e. the least $RPD_{min_{mean}}$, for both 100 and 1000 iterations, only those results are presented in detail (see Tables 4.6 and 4.7 respectively).

All computed solutions of the large instances were feasible. Observe that in the multiple runway cases of problem set "13" the initial solution values even are lower than the best solutions found by [15]. Hence, it can be assumed that the generated initial solutions for both the single and multiple runway cases again provide good starting points for the improvement heuristic. Their computation times are mainly below one second.

Nevertheless, as problem size increases, runtimes beyond one second are accumulated. The highest computation time for building an initial solution was measured for the single runway case of test set “13” with approximately 27 seconds. In this context, recall, that the CPU-time to find an initial solution is mainly determined by the size and characteristics of the LP to be solved.

Concerning the improvement heuristic, in two (using 100 iterations) respectively five (using 1000 iterations) of the 24 instances the runtime limit (per replication) of 7200 seconds (two hours) was exceeded. Nevertheless, VNS was only unable to find an improved solution for problem set “13” involving a single runway. With 100 iterations, the algorithm obtained in all but five of the 24 instances at least the best results found by [15]. For eleven of them it produced even better results. Using 1000 iterations, VNS was able to find in all but four instances at least the best results of [15]. For twelve instances it obtained even better results. Note that this refers also to the average results, especially in the latter case. The largest average computation times of the improvement heuristic for both 100 and 1000 iterations are beyond the defined limit of 7200 seconds. As in those cases also mainly improved results were obtained, the worst average processing times per flight can be indicated as being in an approximate interval of 14 seconds (for problem set “13” involving 500 airplanes) and 48 seconds (for problem set “10” involving 150 airplanes). Finally note that computation times for the single runway cases are in fact significantly higher compared with the multiple runway cases. This is based on the fact that the LPs to be solved are by far more extensive.

Set	*n*	*m*	Z_{best}	Z_{init}	Z_{min}	Z_{max}	Z_{avg}	t_{avg}	*T*	RPD_{min}[14]
9	100	1	5611.70	7310.18	5611.70	5619.15	5612.45	275.05	2751	0
		2	452.92	540.01	448.91	495.77	454.00	21.47	215	-0.89
		3	75.75	89.03	75.75	89.03	78.41	9.58	96	0
		4	0	0	0	0	0	5.50	55	0
10	150	1	12329.31	20142.40	12837.20	13659.10	12972.50	2193.80	21939	4.12
		2	1288.73	1501.74	1179.48	1470.12	1281.48	55.29	553	-8.48
		3	220.79	327.86	230.77	272.34	248.84	25.95	260	4.52
		4	34.22	42.64	34.22	39.42	34.74	10.95	110	0
		5	0	0	0	0	0	7.00	70	0
11	200	1	12418.32	15018.80	12426.40	13088.80	12544.90	3332.61	33328	0.07
		2	1540.84	1742.69	1414.16	1652.8	1544.08	120.14	1202	-8.22
		3	280.82	344.55	253.07	304.16	266.11	41.56	416	-9.88
		4	54.53	54.53	54.53	54.53	54.53	19.18	192	0
		5	0	0	0	0	0	12.21	122	0
12	250	1	16209.78	20145.60	16382.40	17996.60	16721.50	7201.45	72018	1.07*
		2	1961.39	2067.39	1771.04	1968.05	1850.03	211.74	2118	-9.71
		3	290.04	363.23	271.08	350.11	299.60	62.11	622	-6.54
		4	3.49	10.52	2.44	10.52	4.86	29.39	294	-30.09
		5	0	0	0	0	0	18.38	184	0
13	500	1	44832.38	47116.70	47116.70	47116.70	47116.70	7212.05	72147	5.10*
		2	5501.96	4666.6	4212.1	4617.99	4375.71	892.67	8932	-23.44
		3	1108.51	879.86	801.4	879.86	849.81	301.37	3016	-27.71
		4	188.46	154.56	110.76	145.2	127.98	136.69	1369	-41.23
		5	7.35	0	0	0	0	78.17	783	-100
							Mean Value	928.10	9283	-10.47

Table 4.6: Results of large instances with parameter setup "1" and 100 iterations.

[14] * means each run was aborted after 2 hours of computation time.

Set	*n*	*m*	Z_{best}	Z_{init}	Z_{min}	Z_{max}	Z_{avg}	t_{avg}	T	RPD_{min}[15]
9	100	1	5611.70	7310.18	5611.70	5611.70	5611.70	2749.95	27500	0
		2	452.92	540.01	444.10	448.91	445.06	226.00	2260	-1.95
		3	75.75	89.03	75.75	75.75	75.75	91.90	919	0
		4	0	0	0	0	0	59.85	599	0
10	150	1	12329.31	20142.40	12606.70	13132.50	12754.60	7200.38	72005	2.25*
		2	1288.73	1501.74	1149.22	1225.44	1164.46	559.18	5592	-10.83
		3	220.79	327.86	205.21	239.01	208.59	211.07	2111	-7.06
		4	34.22	42.64	34.22	34.22	34.22	107.85	1079	0
		5	0	0	0	0	0	75.85	759	0
11	200	1	12418.32	15018.80	12483.70	13380.80	12639.60	7200.70	72009	0.53*
		2	1540.84	1742.69	1350.53	1417.12	1364.04	1005.09	10051	-12.35
		3	280.82	344.55	253.07	253.15	253.08	370.81	3708	-9.88
		4	54.53	54.53	54.53	54.53	54.53	187.11	1871	0
		5	0	0	0	0	0	122.12	1221	0
12	250	1	16209.78	20145.60	16577.80	17370.40	16806.10	7201.77	72021	2.27*
		2	1961.39	2067.39	1702.10	1749.40	1727.43	1938.05	19381	-13.22
		3	290.04	363.23	245.77	257.61	253.54	642.79	6428	-15.26
		4	3.49	10.52	2.44	10.52	3.25	296.82	2969	-30.09
		5	0	0	0	0	0	186.59	1866	0
13	500	1	44832.38	47116.70	47116.70	47116.70	47116.70	7210.66	72133	5.10*
		2	5501.96	4666.60	4154.26	4480.38	4294.26	7201.00	72015	-24.50*
		3	1108.51	879.86	675.06	762.39	690.16	2983.33	29836	-39.10
		4	188.46	154.56	89.95	107.82	92.83	1423.84	14240	-52.27
		5	7.35	0	0	0	0	765.68	7658	-100
							Mean Value	2084.10	20843	-12.77

Table 4.7: Results of large instances with parameter setup "1" and 1000 iterations.

[15] * means each run was aborted after 2 hours of computation time.

4.2.4.2 Parameter-Specific Aggregated Results

Table 4.8 provides aggregated results of the large instances in dependence of parameter setting and number of iterations.

It shows that runtimes $t_{avg_{mean}}$ and T_{mean} are increasing by unleashing the search, while values of $RPD_{min_{mean}}$ getting worse. Hence, the better choice when dealing with large instances is parameter setup "1" with 100 or 1000 iterations. Note that, if the number of iterations is increased by a factor of "10", mean values of computation times will only be higher by a factor of approximately "2".

Setup	*VNS 100 iterations*			*VNS 1000 iterations*		
	$t_{avg_{mean}}$	T_{mean}	$RPD_{min_{mean}}$	$t_{avg_{mean}}$	T_{mean}	$RPD_{min_{mean}}$
1	928.10	9283	-10.47	2084.10	20843	-12.77
5	1090.46	9741	-8.75	2254.05	22543	-12.42

Table 4.8: Aggregated results of large instances.

4.2.4.3 Summary of Best Solutions

Table 4.9 provides a summary of the best solutions obtained for the large instances, in terms of RPD_{min}, regardless of parameter setup and number of iterations. The respective settings and number of iterations are depicted in the last two columns. If identical results were obtained by different setups, only those requiring the least runtimes are shown.

It is significant that the best solutions were mainly obtained by parameter setup "1". For only four of the 24 large instances, setup "5" was able to find the best result. As can be recognized further, mostly 100 iterations were sufficient to produce excellent results. Without anticipation of the next subchapter, it has to be emphasized that the algorithm found one new best-known solution for test set "11" involving three runways.

Set	*n*	*m*	Z_{min}	Z_{max}	Z_{avg}	t_{avg}	*T*	RPD_{min}[16]	*Setup*	*Iterations*
9	100	1	5611.70	5619.15	5612.45	275.05	2751	0	1	100
		2	444.10	448.91	445.06	226.00	2260	-1.95	1	1000
		3	75.75	89.03	78.41	9.58	96	0	1	100
		4	0	0	0	5.50	55	0	1	100
10	150	1	12606.70	13132.50	12754.60	7200.38	72005	2.25*	1	1000
		2	1149.22	1225.44	1164.46	559.18	5592	-10.83	1	1000
		3	205.21	239.01	208.59	211.07	2111	-7.06	1	1000
		4	34.22	39.42	34.74	10.95	110	0	1	100
		5	0	0	0	7.00	70	0	1	100
11	200	1	12418.30	13261.50	12569.50	5794.03	57942	0	5	100
		2	1350.53	1417.12	1364.04	1005.09	10051	-12.35	1	1000
		3	253.07	304.16	266.11	41.56	416	-9.88	1	100
		4	54.53	54.53	54.53	19.18	192	0	1	100
		5	0	0	0	12.21	122	0	1	100
12	250	1	16382.40	17996.60	16721.50	7201.45	72018	1.07*	1	100
		2	1702.10	1749.40	1727.43	1938.05	19381	-13.22	1	1000
		3	232.99	259.42	238.16	776.50	7766	-19.67	5	1000
		4	2.44	10.52	4.86	29.39	294	-30.09	1	100
		5	0	0	0	18.38	184	0	1	100
13	500	1	47116.70	47116.70	47116.70	7212.05	72147	5.10*	1	100
		2	4037.05	4340.23	4117.81	7200.85	72014	-26.63*	5	1000
		3	673.85	726.25	685.38	3363.63	33639	-39.21	5	1000
		4	89.95	107.82	92.83	1423.84	14240	-52.27	1	1000
		5	0	0	0	78.17	783	-100	1	100

Table 4.9: Summary of best solutions obtained.

[16] * means each run was aborted after 2 hours of computation time.

4.2.5 Comparative Analysis

To analyze the efficiency and effectiveness of the implemented VNS a comparison to existing algorithms is compulsory.

4.2.5.1 Introductory Remarks

In literature, the best results found by [15] during the course of their computational study still provide the benchmark for various recently developed heuristics. This in fact refers to the results of the large instances, as their optimal solutions are not known. Although recently some new best-known solutions were obtained by [20], for reasons of consistency and easier comparability, and because of the fact that [20] is still unpublished[17], the best results found by [15] serve as benchmark in this comparative analysis.

For the time being, only a very small number of papers presented results for all instances of the 13 test sets involving single and multiple runways. In fact, these are only [15] and [20]. The former are the published population based heuristics Scatter Search (SS) and Bionomic Algorithm (BA), the latter is an unpublished Sliding Window Approach (SW), adapted from receding horizon control. These algorithms can be considered as the "state-of-the-art"-heuristics for the underlying problem to solve, both in solution quality as well as in runtimes.

Two VNS heuristics to solve the ALP were also presented recently. A "simple" VNS (VNS-[6]) was developed by [6] and a multi-start adaptive VNS metaheuristic with taboo memory (SVNS-[7]) by [7].[18] Although they only reported results for the small instances, they are of similar type as the implemented VNS. Therefore, a comparison to those algorithms is necessary as well.

With respect to the implemented VNS, for reasons of the comparative study the solutions of the best parameter setup in terms of aggregated results are presented, withstanding the fact that single instances may provide better solutions with a different setting. In other words, it was not the intention to pick out and compare the best solutions obtained (independent of parameter setting), but to compare the performance of the best setup overall. This means, for both 100 and 1000 iterations parameter setup "1" was consulted as it provided the best aggregated results.

---

[17] Recall that results were only presented on "Chinese Control and Decision Conference 2011".

[18] Note that [6] and [7] are also unpublished "conference"-papers.

As the various algorithms were programmed in different languages and the computations are executed on computers with different specifications, runtimes are in fact hard to compare. In detail, SS and BA were implemented in C++ on a 2 GHz Pentium PC with 512 MB of memory, VNS-[6] was programmed in 4D language on a 2 GHz Pentium PC with 512 MB of memory, SVNS-[7] was implemented in 4D language on a 2 GHz Pentium PC with 1 GB of memory, and SW was programmed in Visual C# and run on a 2.6 GHz PC with 1.87 GB of memory. In the absence of accurate measures, a very naive method had to be applied in comparing runtimes of algorithms programmed in different languages and run on different processor types having different sizes of available memory. Under this naive assumption, the 2.8 GHz CPU used for this work can be considered as performing approximately 1.08 times faster than a 2.6 GHz CPU (SW) and roughly 1.4 times faster than a 2 GHz CPU (SS, BA, VNS-[6], SVNS-[7]). Note, in the single-threaded[19] implementation of the underlying algorithm the number of CPU cores of the used processor (in fact a quad-core processor) actually do not play a role, because neither multithreading[20] nor multitasking[21] will be performed.

4.2.5.2 Detailed Results and Interpretation

The comparative table for the small instances had to be split up in two. Table 4.10 provides a comparison of the percentage gaps (from optimal solution) obtained by the different algorithms. Table 4.11 shows their respective runtimes. In Table 4.12 percentage gaps (from best solutions obtained by [15] during their computational work) and computation times of the large instances are summarized.

For reasons of consistency and comparability, as [20] only reported execution times of a single run, in all tables the total computation times for ten replications, as reported by [6], [7] and [15], had to be averaged, i.e. divided by ten and rounded to tenths. Nevertheless, the percentage gaps presented are the best ones found during the course of ten runs. With respect to the following tables, further note that G_{best}, G_{SW} and RPD_{min} have the same meaning.

With respect to the small test sets, the implemented VNS clearly outperforms almost all heuristics in solution quality. Only SW provides the same optimal results for all instances.

[19] Processing of one command at a time.

[20] Form of parallelism of processes.

[21] Another form of parallelism of processes.

With the exception of SW, and when using 100 iterations, the implemented algorithm also performs faster than the other heuristics; different CPU-speeds considered. Nevertheless, when applying 1000 iterations, the algorithm is not competitive anymore.

Referring to the solution qualities of the large test sets, it can be recognized that VNS obtains for all but one instances better solutions than SS and BA. In regard to SW, this is only true for the single runway cases. Nevertheless, when applying 1000 iterations, VNS outperforms the latter in terms of overall solution quality, i.e. $RPD_{min_{mean}}$. Note that this is mainly achieved by providing better results for the single runway instances. In fact, both SW and VNS provide identical solutions for twelve of the 24 instances. For seven of them SW found better results, in five cases VNS obtained the better solutions. For problem set "11" involving three runways, VNS found a new best-known solution.

The major drawback of the implemented VNS can be identified as being the runtimes. Computation times are only competitive for the multiple runway cases when applying 100 iterations. As the runtimes of VNS increase by a factor of roughly "10" when using 1000 instead of 100 iterations, it seems to be clear that these perform the worst.

Set	*m*	*SS*	*BA*	*VNS-[6]*	*SVNS-[7]*	*SW*	*VNS 100*	*VNS 1000*
		G_{best}	G_{best}	G_{best}	G_{best}	G_{SW}	RPD_{min}	RPD_{min}
1	1	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0
	2	0	0	0.91	0	0	0	0
	3	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0
	2	0	0	0.4	0.37	0	0	0
	3	0	0	0	0	0	0	0
	4	0	0	1.7	1.86	0	0	0
5	1	0	0	0	0	0	0	0
	2	0	3.08	3.4	3.72	0	0	0
	3	0	0	0	0	0	0	0
	4	0	0	2.1	1.53	0	0	0
6	1	0	0	0	0	0	0	0
	2	0	3.61	4.6	3.37	0	0	0
	3	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0
	2	0	0	0.4	0.34	0	0	0
8	1	52.05	36.15	41.9	38.64	0	0	0
	2	0	0	0	0	0	0	0
	3	0	0	2.3	0	0	0	0
Mean		2.1	1.7	2.3	1.99	0	0	0

Table 4.10: Comparison of the results of the small instances.

Set	*m*	*SS*	*BA*	*VNS-[6]*	*SVNS-[7]*	*SW*	*VNS 100*	*VNS 1000*
		t_{avg}	t_{avg}	t_{avg}	t_{avg}	T[22]	t_{avg}	t_{avg}
1	1	0.4	6.0	5.7	7.2	0.031	1.50	15.44
	2	2.4	4.5	8.2	9.2	0.032	1.41	14.16
	3	3.9	3.4	4.6	5.6	0.015	1.07	10.56
2	1	0.6	9.0	8.3	9.6	0.046	2.12	21.50
	2	4.5	4.9	5.9	6.4	0.047	1.75	17.16
	3	4.6	4.3	3.5	5.3	0.047	1.34	13.42
3	1	0.8	9.9	8.1	9.8	0.047	2.36	23.55
	2	4.8	5.8	6.7	8.1	0.063	2.44	24.48
	3	6.2	6.3	4.1	7.2	0.047	1.54	15.17
4	1	0.8	9.5	6.7	16.4	0.141	2.52	26.02
	2	5.2	5.5	5.3	9.8	0.375	1.82	18.05
	3	4.6	5.7	7.2	8.3	0.125	1.53	15.29
	4	5.6	5.2	4.3	6.7	0.062	1.38	13.49
5	1	0.9	10.0	7.3	16.1	0.05	2.55	25.50
	2	5.0	6.1	4.7	10.5	0.25	2.38	24.33
	3	5.4	4.3	4.9	6.4	0.187	1.57	15.61
	4	5.6	6.8	8.1	10.3	0.08	1.34	13.45
6	1	15.8	27.4	31.5	35.4	0.047	2.68	26.74
	2	7.0	10.1	16.2	18.4	0.094	2.39	22.31
	3	5.4	8.7	7.9	9.7	0.078	1.22	12.38
7	1	19.5	7.9	12.3	14.2	0.11	6.54	65.84
	2	11.8	12.4	16.3	14.6	0.094	2.96	29.64
8	1	4.2	28.7	16.4	19.7	0.172	12.77	129.17
	2	12.1	19.6	23.8	21.5	0.188	6.51	64.54
	3	13.9	18.1	14.9	17.2	0.15	2.71	27.02
Mean		6.0	9.6	9.7	12.1	0.103	2.74	27.39

Table 4.11: Comparison of the runtimes of the small instances.

[22] As [20] only provided results of a single run, T (total computation time) must be compared with t_{avg}.

Set	*m*	*SS*		*BA*		*SW*		*VNS 100* [23]		*VNS 1000* [24]	
		t_{avg}	G_{best}	t_{avg}	G_{best} [25]	T	G_{SW}	t_{avg}	RPD_{min} [26]	t_{avg}	RPD_{min} [27]
9	1	11.9	30.06	55.4	14.51	0.953	3.58	275.05	0	2749.95	0
	2	34.2	5.67	48.7	54.73	0.515	-1.95	21.47	-0.89	226.00	-1.95
	3	39.0	0	46.6	87.46	0.563	0	9.58	0	91.90	0
	4	33.6	0	43.9	n/d	0.39	0	5.50	0	59.85	0
10	1	22.7	44.96	92.5	33.90	4.047	20.07	2193.8	4.12	7200.38	2.25*
	2	60.8	7.87	84.5	25.95	1.562	-11.25	55.29	-8.48	559.18	-10.83
	3	66.8	8.88	80.3	195.88	1.329	-7.06	25.95	4.52	211.07	-7.06
	4	64.7	16.74	78.8	292.40	0.969	0	10.95	0	107.85	0
	5	60.7	0	76.2	n/d	0.828	0	7.00	0	75.85	0
11	1	25.6	17.95	141.7	16.67	2.454	9.15	3332.61	0.07	7200.70	0.53*
	2	95.9	9.19	128.7	38.54	2.453	-13.62	120.14	-8.22	1005.09	-12.35
	3	102.1	21.59	120.3	290.09	2.75	-9.85	41.56	**-9.88****	370.81	**-9.88****
	4	99.3	2.77	116.8	474.47	1.516	0	19.18	0	187.11	0
	5	95.6	0	115.8	n/d	1.188	0	12.21	0	122.12	0
12	1	38.1	22.15	201.1	23.58	6.219	11.2	7201.45	1.07*	7201.77	2.27*
	2	126.6	18.80	183.5	50.18	8.281	-13.55	211.74	-9.71	1938.05	-13.22
	3	145.4	17.48	171.0	198.01	2.641	-23.47	62.11	-6.54	642.79	-15.26
	4	144.5	271.63	168.8	13216.91	1.906	-30.09	29.39	-30.09	296.82	-30.09
	5	138.6	0	166.2	n/d	1.625	0	18.38	0	186.59	0
13	1	123.7	3.24	585.2	1.03	17.891	-8.49	7212.05	5.10*	7210.66	5.10*
	2	383.6	3.72	537.9	37.47	9.969	-28.52	892.67	-23.44	7201.00	-24.50*
	3	456.0	1.98	515.8	182.69	9.297	-39.21	301.37	-27.71	2983.33	-39.10
	4	441.3	22.98	497.7	1186.81	6.641	-52.27	136.69	-41.23	1423.84	-52.27
	5	442.1	0	488.7	22308.44	5.516	-100	78.17	-100	765.68	-100
Mean		135.5	22.0	197.8	1936.5	3.813	-12.31	928.10	-10.47	2084.10	-12.77

Table 4.12: Comparison of the results and runtimes of the large instances.

[23] Parameter setup "1" adopted.

[24] Parameter setup "1" adopted.

[25] n/d means not defined.

[26] * means each run was aborted after 2 hours of computation time; ** means new best-known result found.

[27] * means each run was aborted after 2 hours of computation time; ** means new best-known result found.

4.2.5.3 Summary of Comparative Study

To briefly summarize, the implemented VNS proofed to provide excellent results for all of the 13 test sets, involving single and multiple runways. Nonetheless, the main drawback lies in its computation time, especially as the problem size increases. This seems to be an evidence of poor handling of large sized instances. In turn, this is mainly influenced by solving a lot of LPs involving many variables and constraints. Nevertheless, as the main concern of this work lies on solution quality rather than computation time, the developed metaheuristic proofed to perform satisfactorily.

Finally, the outstanding low computation times of SW paired with excellent results for all of the 13 datasets must be emphasized.

4.3 Real World Problem

4.3.1 Introductory Remarks

The real world scenario is based on actual flight events occurred at Munich Airport (MUC) on Thursday 28th July 2011. As this airport consists of two parallel runways capable of independent runway operations, it fits perfectly into the framework tested on the instances in the previous section. Necessary data was available from [9]. Data not provided by officially published sources were assumed.

This case study does not represent a snapshot of arriving and departing flights at a specific point in time. Rather, all times required were derived post-priori (under various modeling assumptions) from manually recorded real estimated times of arrival respectively departure (if they were published). In other words, each flight having an assumed target take-off or landing time between 10.00 am and 11.00 am was considered. The term “assumed” refers to the fact that real target times of airplanes involved are actually not known.[28] Nevertheless, within this one hour time frame, 22 departures and 38 arrivals (60 flights in total) had to be managed, i.e. allocated to a runway, sequenced and scheduled. In fact, this reflects a dense traffic situation which usually occurs several times a day at MUC.

In the underlying scenario, a dual mixed independent runway operation under normal weather conditions, i.e. no adverse weather situations (e.g. fog, thunderstorms, etc.) requiring increased separations, was supposed.

[28] Recall that target times are solely related to airplanes.

4.3.2 Modeling of Input Data

In the absence of more detailed input data, several modeling assumptions have to be made. Basically, these assumptions are consistent with those made in [2] and [3].

Target landing times were derived from manually recorded real estimated on-block times, i.e. arrival times at the gate, assuming a uniform taxi-in time[29] of five minutes.

As expected off-block times, i.e. departure times from the gate, are usually not published, target take-off times were estimated assuming all departures leave the terminal on-time and having a uniform taxi-out time[30] of 15 minutes. If an estimated off-block time was actually published, this was interpreted as due to an allocated take-off time (CTOT). In this special case, the CTOT was assumed to represent the target take-off time, and was supposed to be 15 minutes after the published estimated off-block time.

Earliest landing and take-off times were randomly generated to be between one and ten minutes prior to their respective target times. Latest landing and take-off times were assumed to be 30 minutes after their respective earliest times. In the presence of a CTOT, the earliest and latest take-off times are defined by convention to be five minutes before and ten minutes after the target time (which was assumed to equal the CTOT) respectively.

Marginal penalty costs for being before or after target time were generated as real numbers from the interval [1, 2].

Minimum separation times were computed in accordance with [12] (see Table 4.13). Any minimum separation distance was converted into its time-equivalent assuming a uniform approach speed of 160 KTS for all airplanes involved.

			Predecessor			
			Departure		*Arrival*	
			Heavy	*Medium*	*Heavy*	*Medium*
Successor	*Departure*	*Heavy*	60	60	50	50
		Medium	120	60	50	50
	Arrival	*Heavy*	60	60	90	68
		Medium	60	60	113	68

Table 4.13: Minimum separation times between airplanes in seconds.[31]

[29] Necessary time from the landing runway to the parking position.

[30] Necessary time from the parking position to the departure runway.

[31] Wake turbulence category: Medium/Heavy: Maximum Certified Take-Off Mass below/above 136 t.

The generated input data is presented in Appendix A. The minimum separation matrix of the real world instance is asymmetrical and does not fulfill the triangle inequality. This is actually not a surprise as a mix of departing and arriving airplanes, belonging to different wake turbulence categories, is considered. Compared to the test instances, the real world instance shows a higher traffic density. In addition, time windows of examined flights are fairly narrow.

4.3.3 Results of the Real World Problem

Appendix A shows details concerning runway allocation, sequence of flights and scheduled times of the initial and best solutions. Thereby, parameter setup "1" with 100 and 1000 iterations was applied. As usual in a static situation, the depicted sequences and schedules must be seen as to be in an enclosed world, not encountering any conflicting traffic before and after the time frame examined.

The computational results for all six parameter setups (with 100 and 1000 iterations) are reported in Table 4.14. For the initial solution, an evaluation value of Z_{init} = 2918.69 was obtained within a fraction of a second.

Setup	*VNS 100*				*VNS 1000*			
	Z_{min}	Z_{max}	Z_{avg}	t_{avg}	Z_{min}	Z_{max}	Z_{avg}	t_{avg}
1	2125	2286.56	2146.76	13.48	2140.80	2143.41	2141.06	162.37
2	2125	2174.97	2139.57	16.86	2130.56	2130.56	2130.56	219.24
3	2131.27	2479.29	2222.89	17.54	2125	2125	2125	197.96
4	2125	2237.08	2141.29	21.03	2125	2125	2125	210.63
5	2125	2177.68	2135.52	20.68	2125	2125	2125	216.50
6	2125	2151.16	2128.71	20.27	2125	2125	2125	227.73

Table 4.14: Results of the real world scenario.

The least evaluation value obtained equals 2125. As the optimal evaluation value of the real world scenario is not known, this corresponds to the best-known solution. Although it was found with each setup, stable results actually require parameter settings "3" to "6" paired with 1000 iterations. The average runtimes in fact are higher compared with the test instances of similar size. Obviously, the denser traffic situation and narrower time

windows require a higher number of less restrictive random shaking steps, while making it more difficult for the LP-solver to find an optimal schedule of a specific sequence.

4.3.4 Comparative Analysis

To emphasize the effectiveness of optimized runway operations scheduling, the following hypothetical comparative analysis was performed. For this reason, all flights in the underlying scenario were sequenced according to what can be interpreted as a “first-come first-serve”-methodology. In fact, this strategy is still prevailing at the majority of large international airports.

In detail, airplanes were sorted by their non-decreasing earliest times, and then assigned to the respective runways in the same manner as introduced, i.e. by the round robin procedure. Anyway, no further optimization of runway sequencing was performed, i.e. the initial runway allocation and the resulting sequences were frozen. Of course, it seems to be self-explanatory that earliest times used for allocation and sequencing must be based on accurate estimates. In other words, the freezing of runway assignment and sequence has to occur at a late stage, e.g. upon reaching the radar range of the respective air traffic approach control. The schedule was computed under consideration of individual target times, not to unnecessarily speed-up or slow-down any flight.

Under these assumptions, costs of 7363.02 were incurred. Related to the best solution found above, this corresponds to additional costs of 5238.02. Even if compared to the worst solution obtained, the cost increase still equals 4883.73. In other words, in this scenario optimized runway operations scheduling would gain cost savings in the range of roughly 65-70 %.

In fact, similar outputs may be encountered in daily air traffic operation at any airport worldwide. Although costs are of pure fictitious, this is a proof of efficient runway operations scheduling to be of utmost importance when managing air traffic.

Finally, it must be emphasized that the assumptions made in this comparative analysis are pure hypothetical and does not reflect the way air traffic management is performed at Munich International Airport.

5. Conclusion and Future Research

This master thesis dealt with the single and multiple runway cases of the Aircraft Sequencing Problem (ASP) involving departing and arriving airplanes. Following a number of authors, the ASP was formulated as static decision problem exclusively considering immediate runway traffic. The linear objective function adopted corresponds to the minimization of total weighted deviation from individual target times as originally introduced by [1].

In the underlying work, the ASP was considered as routing problem. A Variable Neighborhood Search (VNS), basically designed for problems of transportation logistics, was presented as solution method. To adapt the VNS to the problem considered, problem-specific knowledge had to be incorporated.

To measure the performance of the implemented metaheuristic, 13 test sets provided by the OR-library of J.E. Beasley were adopted. Computational results for all 49 test instances, involving from 10 to 500 airplanes, in the presence of single and multiple runways were presented. Additionally, a comparative analysis was performed with current "state-of-the-art" heuristics. The implemented VNS provided excellent results, outperforming all but one heuristics previously presented. Nevertheless, drawbacks regarding runtimes exist, especially as problem size increases.

Finally, the application of the algorithm to a real world scenario encountered at Munich International Airport was examined. A comparative analysis with a "first-come first-serve"-methodology reveals potential cost savings up to approximately 70%. This again emphasizes the effectiveness of optimized runway operations management.

Regarding future research, the primary task consists in reducing the runtimes of the VNS by applying some algorithmic modifications. A possibility would be to combine the implemented VNS with a sliding window approach similar to the one used by [20], i.e. to embed the metaheuristic in a rolling horizon framework. In other words, the total problem will be divided into a number of small problems that will be solved one after the other. In fact, this is a very simply approach and, if properly tuned, is capable of reducing runtimes significantly.

It is further of special interest to see how VNS will then perform in a dynamic environment where new airplanes appear and others disappear on the planning horizon, and how it will handle additional real world constraints.

Another very interesting feature would be the implementation of a different objective function. In order to handle increased air traffic demand, additional runway capacity could be gained by considering the problem from the viewpoint of Air Traffic Control (ATC). This was already examined in various real world case studies. With respect to the results obtained in this paper, expectations to perform satisfactorily are actually very high.

Bibliography

[1] **Beasley, J.E., Krishnamoorthy, M., Sharaiha, Y.M., Abramson, D.:** Scheduling Aircraft Landings – The Static Case. *Transportation Science*, 34(2):180-197, 2000.

[2] **Beasley, J.E., Sonander, J., Havelock, P.:** Scheduling aircraft landings at London Heathrow using a population heuristic. *Journal of the Operational Research Society*, 52(5):483-493, 2001.

[3] **Beasley, J.E., Krishnamoorthy, M., Sharaiha, Y.M., Abramson, D.:** Displacement problem and dynamically scheduling aircraft landings. *Journal of the Operational Research Society*, 55(1):54-64, 2004.

[4] **Beasley, J.E.:** *OR-Library*. Last accessed on 30th September 2011, available from: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/airlandinfo.html

[5] **Bianco, L., Dell´Olmo, P., Giordani, S.:** Minimizing total completion time subject to release dates and sequence-dependent processing times. *Annals of Operations Research*, 86(0):393-415, 1999.

[6] **Dhouib, S.:** A Variable Neighborhood Search Metaheuristic for Multiple Runways Aircraft Landing Problem. In *International Conference on Metaheuristics and Nature Inspired Computing (META 2010), Djerba Island, Tunisia, 27th-31th October 2010*, 2010.

[7] **Dhouib, S.:** A Multi Start Adaptive Variable Neighborhood Search Metaheuristic for the Aircraft Landing Problem. In *4th International Conference on Logistics (LOGISTIQUA 2011), Hammamet, Tunisia, 31th May-3rd June 2011*, pages 197-200, 2011.

[8] **Fahle, T., Feldmann, R., Götz, S., Grothklags, S., Monien, B.:** The Aircraft Sequencing Problem. In R. Klein et al., editors, *Computer Science in Perspective. Essays dedicated to Thomas Ottmann*, Lecture Notes in Computer Science 2598, pages 152-166, Springer-Verlag, Berlin Heidelberg, 2003.

[9] **FlightStats Homepage:** Last accessed on 30th September 2011, available from: http://www.flightstats.com

[10] **Hansen, P., Mladenović, N.:** A Tutorial on Variable Neighborhood Search. Technical Report G-2003-46, Les Cahiers du GERAD, HEC Montréal and GERAD, Canada, 2003.

[11] **Hemmelmayr, V., Doerner, K., Hartl, R.F.:** A Variable Neighborhood Search Heuristic for Periodic Routing Problems. *European Journal of Operations Research*, 195(3):791-802, 2009.

[12] **ICAO:** *Procedures for Air Navigation Services: Air Traffic Management (PANS-ATM, Doc. 4444)*, ICAO, 15th edition, 2007.

[13] **lp_solve Homepage:** Last accessed on 30th September 2011, available from: http://lpsolve.sourceforge.net/

[14] **Mladenović, N., Hansen, P.:** Variable Neighborhood Search. *Computers and Operations Research*, 24(11):1097-1100, 1997.

[15] **Pinol, H., Beasley, J.E.:** Scatter Search and Bionomic Algorithms for the aircraft landing problem. *European Journal of Operations Research*, 171(2):439-462, 2006.

[16] **Pirkwieser, S., Raidl, G.R.:** A Variable Neighborhood Search for the Periodic Vehicle Routing Problem with Time Windows. In C. Prodhon et al., editors, *Proceedings of the 9th EU/MEeting on Metaheuristics for Logistics and Vehicle Routing, Troyes, France, 23rd -24th October 2008*, 2008.

[17] **Polacek, M., Hartl, R.F., Doerner, K., Reimann, M.:** A Variable Neighborhood Search for the Multi Depot Vehicle Routing Problem with Time Windows. *Journal of Heuristics*, 10(6):613-627, 2004.

[18] **Polacek, M.:** *Variable Neighborhood Search for Routing Problems.* Doctoral thesis, University of Vienna, Faculty of Business, Economics and Statistics, 2008.

[19] **Soomer, M.J.:** *Runway Operations Scheduling using Airline Preferences*. PhD thesis, Vrije Universiteit Amsterdam, Faculteit der Exacte Wetenschappen, 2009.

[20] **Xiangwei M., Ping Z., Chunjin L.:** Sliding Window Algorithm for Aircraft Landing Problem. In *Chinese Control and Decision Conference (2011 CCDC), Mianyang, China, 23rd-25th May 2011*, pages 874-879, 2011.

Appendix A: Detailed Results of Real World Problem

The following notation is used in the tables of this appendix:

ID Airplane identification as used by data structures

Flight Nr. Commercial flight number according officially published time-table

Phase Flight phase: D = Departure, A = Arrival

Cat. Wake turbulence category: M = Medium, H = Heavy

ST Scheduled time as computed by the program

EGT Estimated gate time: the time a flight was expected to arrive at the gate or to leave the gate

SLT Calculated Take-Off Time (CTOT)

TT Target time: derived from *EGT*:

Arrivals: $TT = EGT - 5$ minutes

Departures: $TT = EGT + 15$ minutes

In the case of CTOT (only for departures): $TT = SLT$

ET, LT Earliest time, Latest time: derived from *TT*:

$ET = TT$ – random [1, 10]

$LT = ET + 30$ minutes

In the case of CTOT (only for departures): $ET = TT - 5$ minutes

$LT = TT + 10$ minutes

EP, LP Marginal early or late penalty cost (per unit of time) for being scheduled before or after target time: randomly determined in the interval [1, 2].

GENERATED INPUT DATA

Departures					
ID	*Flight Nr.*	*Phase*	*Cat.*	*EGT/SLT/ET/TT/LT*	*EP/LP*
0	DL131	D	H	0950/1015/1010/1015/1025	1.74744/1.55548
1	OU4433	D	M	1000/0/1013/1015/1043	1.68909/1.72836
2	LH107	D	M	1000/1040/1035/1040/1050	1.25311/1.75067
3	U25382	D	M	1010/0/1019/1025/1049	1.12692/1.31406
4	OU4439	D	M	1015/0/1025/1030/1055	1.68756/1.1145
5	AB6300	D	M	1015/0/1023/1030/1053	1.44608/1.03506
6	AB6026	D	M	1020/0/1025/1035/1055	1.82254/1.55252
7	AY804	D	M	1020/0/1032/1035/1102	1.2627/1.01184
8	A3501	D	M	1020/0/1028/1035/1058	1.20044/1.36356
9	RO316	D	M	1025/0/1035/1040/1105	1.52249/1.39545
10	TK1630	D	M	1030/0/1043/1045/1113	1.69086/1.98947
11	LY354	D	M	1030/0/1037/1045/1107	1.18015/1.43527
12	AB6188	D	M	1030/0/1043/1045/1113	1.38522/1.33182
13	AB8282	D	M	1035/1055/1050/1055/1105	1.15308/1.56119
14	LH2230	D	M	1040/0/1048/1055/1118	1.95837/1.04385
15	LH2066	D	M	1040/0/1048/1055/1118	1.73782/1.30496
16	EN1950	D	M	1045/0/1055/1100/1125	1.90781/1.79709
17	EN1896	D	M	1045/0/1058/1100/1128	1.41647/1.51028
18	CL2272	D	M	1045/0/1052/1100/1122	1.93961/1.0094
19	CL2442	D	M	1045/0/1057/1100/1127	1.25046/1.94989
20	EN1956	D	M	1045/0/1056/1100/1126	1.66946/1.35324
21	LH2286	D	M	1045/1115/1110/1115/1125	1.38727/1.20053

Arrivals					
ID	*Flight Nr.*	*Phase*	*Cat.*	*EGT/SLT/ET/TT/LT*	*EP/LP*
22	CL1853	A	M	1009/0/1003/1004/1033	1.55804/1.82623
23	LH2367	A	M	1006/0/952/1001/1022	1.60287/1.63214
24	AC846	A	H	1015/0/1001/1010/1031	1.48511/1.5351
25	CL2271	A	M	1005/0/957/1000/1027	1.07251/1.30954
26	CL1675	A	M	1005/0/950/1000/1020	1.88263/1.97531
27	CL2227	A	M	1024/0/1009/1019/1039	1.98709/1.71082
28	CL2319	A	M	1005/0/950/1000/1020	1.72968/1.323
29	IQ1689	A	M	1005/0/955/1000/1025	1.41559/1.80487
30	CL2301	A	M	1037/0/1029/1032/1059	1.62744/1.01266
31	CL2441	A	M	1010/0/1000/1005/1030	1.32452/1.99289
32	LH2483	A	M	1025/0/1012/1020/1042	1.38144/1.99124
33	CL1841	A	M	1015/0/1007/1010/1037	1.72256/1.67462
34	CL2247	A	M	1010/0/1003/1005/1033	1.70837/1.5968
35	EN2341	A	M	1005/0/957/1000/1027	1.93002/1.43997
36	LH1817	A	M	1015/0/1003/1010/1033	1.06827/1.06262
37	CL2135	A	M	1025/0/1012/1020/1042	1.97336/1.88531
38	LH1983	A	M	1020/0/1007/1015/1037	1.62146/1.64594
39	AB6193	A	M	1025/0/1018/1020/1048	1.10474/1.05487
40	CL2261	A	M	1023/0/1008/1018/1038	1.31061/1.38208
41	CL2175	A	M	1040/0/1031/1035/1101	1.44662/1.50711
42	IQ1621	A	M	1015/0/1003/1010/1033	1.06775/1.4631
43	LH2031	A	M	1025/0/1018/1020/1048	1.76123/1.00626
44	LH2003	A	M	1020/0/1011/1015/1041	1.14639/1.24191
45	IQ2393	A	M	1020/0/1006/1015/1036	1.345/1.83859
46	CL1611	A	M	1016/0/1002/1011/1032	1.57843/1.86105
47	LH2061	A	M	1020/0/1009/1015/1039	1.04828/1.75391
48	KF789	A	M	1020/0/1006/1015/1036	1.75623/1.50668
49	IQ2155	A	M	1035/0/1025/1030/1055	1.41977/1.53189
50	V3321	A	M	1040/0/1027/1035/1057	1.13971/1.28735
51	AB9721	A	M	1035/0/1022/1030/1052	1.97607/1.17358
52	CL2491	A	M	1045/0/1030/1040/1100	1.08829/1.73813
53	BA948	A	M	1045/0/1037/1040/1107	1.59619/1.71768
54	AZ432	A	M	1050/0/1041/1045/1111	1.81793/1.94928
55	U23411	A	M	1050/0/1043/1045/1113	1.04108/1.68204
56	WA1793	A	M	1055/0/1041/1050/1111	1.48325/1.32034
57	EN1917	A	M	1035/0/1022/1030/1052	1.59824/1.13007
58	LH106	A	M	1100/0/1048/1055/1118	1.38446/1.93893
59	LH2033	A	M	1100/0/1045/1055/1115	1.85117/1.36209

INITIAL SOLUTION

Runway 0						
ID	*Flight Nr.*	*Phase*	*Cat.*	*ST*	*EGT/SLT/ET/TT/LT*	*EP/LP*
25	CL2271	A	M	958	1005/0/957/1000/1027	1.07251/1.30954
28	CL2319	A	M	1000	1005/0/950/1000/1020	1.72968/1.323
35	EN2341	A	M	1001	1005/0/957/1000/1027	1.93002/1.43997
22	CL1853	A	M	1003	1009/0/1003/1004/1033	1.55804/1.82623
34	CL2247	A	M	1005	1010/0/1003/1005/1033	1.70837/1.5968
33	CL1841	A	M	1010	1015/0/1007/1010/1037	1.72256/1.67462
42	IQ1621	A	M	1011	1015/0/1003/1010/1033	1.06775/1.4631
0	DL131	D	H	1014	950/1015/1010/1015/1025	1.74744/1.55548
38	LH1983	A	M	1015	1020/0/1007/1015/1037	1.62146/1.64594
45	IQ2393	A	M	1016	1020/0/1006/1015/1036	1.345/1.83859
48	KF789	A	M	1017	1020/0/1006/1015/1036	1.75623/1.50668
27	CL2227	A	M	1018	1024/0/1009/1019/1039	1.98709/1.71082
37	CL2135	A	M	1020	1025/0/1012/1020/1042	1.97336/1.88531
43	LH2031	A	M	1021	1025/0/1018/1020/1048	1.76123/1.00626
4	OU4439	D	M	1029	1015/0/1025/1030/1055	1.68756/1.1145
49	IQ2155	A	M	1030	1035/0/1025/1030/1055	1.41977/1.53189
57	EN1917	A	M	1031	1035/0/1022/1030/1052	1.59824/1.13007
6	AB6026	D	M	1034	1020/0/1025/1035/1055	1.82254/1.55252
8	A3501	D	M	1035	1020/0/1028/1035/1058	1.20044/1.36356
50	V3321	A	M	1036	1040/0/1027/1035/1057	1.13971/1.28735
9	RO316	D	M	1039	1025/0/1035/1040/1105	1.52249/1.39545
53	BA948	A	M	1040	1045/0/1037/1040/1107	1.59619/1.71768
11	LY354	D	M	1044	1030/0/1037/1045/1107	1.18015/1.43527
54	AZ432	A	M	1045	1050/0/1041/1045/1111	1.81793/1.94928
56	WA1793	A	M	1050	1055/0/1041/1050/1111	1.48325/1.32034
14	LH2230	D	M	1055	1040/0/1048/1055/1118	1.95837/1.04385
58	LH106	A	M	1056	1100/0/1048/1055/1118	1.38446/1.93893
16	EN1950	D	M	1059	1045/0/1055/1100/1125	1.90781/1.79709
18	CL2272	D	M	1100	1045/0/1052/1100/1122	1.93961/1.0094
20	EN1956	D	M	1101	1045/0/1056/1100/1126	1.66946/1.35324

Runway 1						
ID	*Flight Nr.*	*Phase*	*Cat.*	*ST*	*EGT/SLT/ET/TT/LT*	*EP/LP*
26	CL1675	A	M	958	1005/0/950/1000/1020	1.88263/1.97531
29	IQ1689	A	M	1000	1005/0/955/1000/1025	1.41559/1.80487
23	LH2367	A	M	1001	1006/0/952/1001/1022	1.60287/1.63214
31	CL2441	A	M	1005	1010/0/1000/1005/1030	1.32452/1.99289
24	AC846	A	H	1008	1015/0/1001/1010/1031	1.48511/1.5351
36	LH1817	A	M	1010	1015/0/1003/1010/1033	1.06827/1.06262
46	CL1611	A	M	1011	1016/0/1002/1011/1032	1.57843/1.86105
1	OU4433	D	M	1014	1000/0/1013/1015/1043	1.68909/1.72836
44	LH2003	A	M	1015	1020/0/1011/1015/1041	1.14639/1.24191
47	LH2061	A	M	1016	1020/0/1009/1015/1039	1.04828/1.75391
40	CL2261	A	M	1018	1023/0/1008/1018/1038	1.31061/1.38208
32	LH2483	A	M	1020	1025/0/1012/1020/1042	1.38144/1.99124
39	AB6193	A	M	1021	1025/0/1018/1020/1048	1.10474/1.05487
3	U25382	D	M	1025	1010/0/1019/1025/1049	1.12692/1.31406
5	AB6300	D	M	1029	1015/0/1023/1030/1053	1.44608/1.03506
51	AB9721	A	M	1030	1035/0/1022/1030/1052	1.97607/1.17358
30	CL2301	A	M	1032	1037/0/1029/1032/1059	1.62744/1.01266
7	AY804	D	M	1034	1020/0/1032/1035/1102	1.2627/1.01184
41	CL2175	A	M	1035	1040/0/1031/1035/1101	1.44662/1.50711
2	LH107	D	M	1039	1000/1040/1035/1040/1050	1.25311/1.75067
52	CL2491	A	M	1040	1045/0/1030/1040/1100	1.08829/1.73813
10	TK1630	D	M	1044	1030/0/1043/1045/1113	1.69086/1.98947
12	AB6188	D	M	1045	1030/0/1043/1045/1113	1.38522/1.33182
55	U23411	A	M	1046	1050/0/1043/1045/1113	1.04108/1.68204
13	AB8282	D	M	1054	1035/1055/1050/1055/1105	1.15308/1.56119
15	LH2066	D	M	1055	1040/0/1048/1055/1118	1.73782/1.30496
59	LH2033	A	M	1056	1100/0/1045/1055/1115	1.85117/1.36209
17	EN1896	D	M	1059	1045/0/1058/1100/1128	1.41647/1.51028
19	CL2442	D	M	1100	1045/0/1057/1100/1127	1.25046/1.94989
21	LH2286	D	M	1115	1045/1115/1110/1115/1125	1.38727/1.20053

BEST SOLUTION WITH 100 ITERATIONS

Runway 0						
ID	*Flight Nr.*	*Phase*	*Cat.*	*ST*	*EGT/SLT/ET/TT/LT*	*EP/LP*
25	CL2271	A	M	958	1005/0/957/1000/1027	1.07251/1.30954
35	EN2341	A	M	1000	1005/0/957/1000/1027	1.93002/1.43997
23	LH2367	A	M	1001	1006/0/952/1001/1022	1.60287/1.63214
22	CL1853	A	M	1003	1009/0/1003/1004/1033	1.55804/1.82623
34	CL2247	A	M	1005	1010/0/1003/1005/1033	1.70837/1.5968
42	IQ1621	A	M	1008	1015/0/1003/1010/1033	1.06775/1.4631
33	CL1841	A	M	1010	1015/0/1007/1010/1037	1.72256/1.67462
46	CL1611	A	M	1011	1016/0/1002/1011/1032	1.57843/1.86105
38	LH1983	A	M	1014	1020/0/1007/1015/1037	1.62146/1.64594
0	DL131	D	H	1015	950/1015/1010/1015/1025	1.74744/1.55548
44	LH2003	A	M	1016	1020/0/1011/1015/1041	1.14639/1.24191
27	CL2227	A	M	1018	1024/0/1009/1019/1039	1.98709/1.71082
37	CL2135	A	M	1020	1025/0/1012/1020/1042	1.97336/1.88531
43	LH2031	A	M	1021	1025/0/1018/1020/1048	1.76123/1.00626
49	IQ2155	A	M	1029	1035/0/1025/1030/1055	1.41977/1.53189
4	OU4439	D	M	1030	1015/0/1025/1030/1055	1.68756/1.1145
57	EN1917	A	M	1031	1035/0/1022/1030/1052	1.59824/1.13007
50	V3321	A	M	1034	1040/0/1027/1035/1057	1.13971/1.28735
6	AB6026	D	M	1035	1020/0/1025/1035/1055	1.82254/1.55252
52	CL2491	A	M	1039	1045/0/1030/1040/1100	1.08829/1.73813
2	LH107	D	M	1040	1000/1040/1035/1040/1050	1.25311/1.75067
11	LY354	D	M	1044	1030/0/1037/1045/1107	1.18015/1.43527
54	AZ432	A	M	1045	1050/0/1041/1045/1111	1.81793/1.94928
12	AB6188	D	M	1045	1030/0/1043/1045/1113	1.38522/1.33182
56	WA1793	A	M	1050	1055/0/1041/1050/1111	1.48325/1.32034
58	LH106	A	M	1055	1100/0/1048/1055/1118	1.38446/1.93893
14	LH2230	D	M	1055	1040/0/1048/1055/1118	1.95837/1.04385
19	CL2442	D	M	1059	1045/0/1057/1100/1127	1.25046/1.94989
16	EN1950	D	M	1100	1045/0/1055/1100/1125	1.90781/1.79709
18	CL2272	D	M	1101	1045/0/1052/1100/1122	1.93961/1.0094

Runway 1						
ID	*Flight Nr.*	*Phase*	*Cat.*	*ST*	*EGT/SLT/ET/TT/LT*	*EP/LP*
29	IQ1689	A	M	958	1005/0/955/1000/1025	1.41559/1.80487
26	CL1675	A	M	1000	1005/0/950/1000/1020	1.88263/1.97531
28	CL2319	A	M	1001	1005/0/950/1000/1020	1.72968/1.323
31	CL2441	A	M	1005	1010/0/1000/1005/1030	1.32452/1.99289
36	LH1817	A	M	1008	1015/0/1003/1010/1033	1.06827/1.06262
24	AC846	A	H	1010	1015/0/1001/1010/1031	1.48511/1.5351
47	LH2061	A	M	1013	1020/0/1009/1015/1039	1.04828/1.75391
45	IQ2393	A	M	1014	1020/0/1006/1015/1036	1.345/1.83859
1	OU4433	D	M	1015	1000/0/1013/1015/1043	1.68909/1.72836
48	KF789	A	M	1016	1020/0/1006/1015/1036	1.75623/1.50668
40	CL2261	A	M	1018	1023/0/1008/1018/1038	1.31061/1.38208
32	LH2483	A	M	1020	1025/0/1012/1020/1042	1.38144/1.99124
39	AB6193	A	M	1021	1025/0/1018/1020/1048	1.10474/1.05487
3	U25382	D	M	1025	1010/0/1019/1025/1049	1.12692/1.31406
51	AB9721	A	M	1030	1035/0/1022/1030/1052	1.97607/1.17358
5	AB6300	D	M	1030	1015/0/1023/1030/1053	1.44608/1.03506
30	CL2301	A	M	1032	1037/0/1029/1032/1059	1.62744/1.01266
8	A3501	D	M	1034	1020/0/1028/1035/1058	1.20044/1.36356
41	CL2175	A	M	1035	1040/0/1031/1035/1101	1.44662/1.50711
7	AY804	D	M	1035	1020/0/1032/1035/1102	1.2627/1.01184
53	BA948	A	M	1040	1045/0/1037/1040/1107	1.59619/1.71768
9	RO316	D	M	1040	1025/0/1035/1040/1105	1.52249/1.39545
55	U23411	A	M	1044	1050/0/1043/1045/1113	1.04108/1.68204
10	TK1630	D	M	1045	1030/0/1043/1045/1113	1.69086/1.98947
13	AB8282	D	M	1054	1035/1055/1050/1055/1105	1.15308/1.56119
59	LH2033	A	M	1055	1100/0/1045/1055/1115	1.85117/1.36209
15	LH2066	D	M	1055	1040/0/1048/1055/1118	1.73782/1.30496
17	EN1896	D	M	1100	1045/0/1058/1100/1128	1.41647/1.51028
20	EN1956	D	M	1101	1045/0/1056/1100/1126	1.66946/1.35324
21	LH2286	D	M	1115	1045/1115/1110/1115/1125	1.38727/1.20053

BEST SOLUTION WITH 1000 ITERATIONS

Runway 0						
ID	*Flight Nr.*	*Phase*	*Cat.*	*ST*	*EGT/SLT/ET/TT/LT*	*EP/LP*
25	CL2271	A	M	958	1005/0/957/1000/1027	1.07251/1.30954
35	EN2341	A	M	1000	1005/0/957/1000/1027	1.93002/1.43997
28	CL2319	A	M	1001	1005/0/950/1000/1020	1.72968/1.323
22	CL1853	A	M	1003	1009/0/1003/1004/1033	1.55804/1.82623
34	CL2247	A	M	1005	1010/0/1003/1005/1033	1.70837/1.5968
42	IQ1621	A	M	1008	1015/0/1003/1010/1033	1.06775/1.4631
24	AC846	A	H	1010	1015/0/1001/1010/1031	1.48511/1.5351
45	IQ2393	A	M	1014	1020/0/1006/1015/1036	1.345/1.83859
0	DL131	D	H	1015	950/1015/1010/1015/1025	1.74744/1.55548
48	KF789	A	M	1016	1020/0/1006/1015/1036	1.75623/1.50668
27	CL2227	A	M	1018	1024/0/1009/1019/1039	1.98709/1.71082
37	CL2135	A	M	1020	1025/0/1012/1020/1042	1.97336/1.88531
43	LH2031	A	M	1021	1025/0/1018/1020/1048	1.76123/1.00626
49	IQ2155	A	M	1029	1035/0/1025/1030/1055	1.41977/1.53189
4	OU4439	D	M	1030	1015/0/1025/1030/1055	1.68756/1.1145
57	EN1917	A	M	1031	1035/0/1022/1030/1052	1.59824/1.13007
50	V3321	A	M	1034	1040/0/1027/1035/1057	1.13971/1.28735
6	AB6026	D	M	1035	1020/0/1025/1035/1055	1.82254/1.55252
53	BA948	A	M	1040	1045/0/1037/1040/1107	1.59619/1.71768
9	RO316	D	M	1040	1025/0/1035/1040/1105	1.52249/1.39545
54	AZ432	A	M	1045	1050/0/1041/1045/1111	1.81793/1.94928
12	AB6188	D	M	1045	1030/0/1043/1045/1113	1.38522/1.33182
56	WA1793	A	M	1050	1055/0/1041/1050/1111	1.48325/1.32034
58	LH106	A	M	1055	1100/0/1048/1055/1118	1.38446/1.93893
14	LH2230	D	M	1055	1040/0/1048/1055/1118	1.95837/1.04385
17	EN1896	D	M	1100	1045/0/1058/1100/1128	1.41647/1.51028
20	EN1956	D	M	1101	1045/0/1056/1100/1126	1.66946/1.35324

Runway 1						
ID	*Flight Nr.*	*Phase*	*Cat.*	*ST*	*EGT/SLT/ET/TT/LT*	*EP/LP*
29	IQ1689	A	M	958	1005/0/955/1000/1025	1.41559/1.80487
26	CL1675	A	M	1000	1005/0/950/1000/1020	1.88263/1.97531
23	LH2367	A	M	1001	1006/0/952/1001/1022	1.60287/1.63214
31	CL2441	A	M	1005	1010/0/1000/1005/1030	1.32452/1.99289
36	LH1817	A	M	1008	1015/0/1003/1010/1033	1.06827/1.06262
33	CL1841	A	M	1010	1015/0/1007/1010/1037	1.72256/1.67462
46	CL1611	A	M	1011	1016/0/1002/1011/1032	1.57843/1.86105
47	LH2061	A	M	1013	1020/0/1009/1015/1039	1.04828/1.75391
44	LH2003	A	M	1014	1020/0/1011/1015/1041	1.14639/1.24191
1	OU4433	D	M	1015	1000/0/1013/1015/1043	1.68909/1.72836
38	LH1983	A	M	1016	1020/0/1007/1015/1037	1.62146/1.64594
40	CL2261	A	M	1018	1023/0/1008/1018/1038	1.31061/1.38208
32	LH2483	A	M	1020	1025/0/1012/1020/1042	1.38144/1.99124
39	AB6193	A	M	1021	1025/0/1018/1020/1048	1.10474/1.05487
3	U25382	D	M	1025	1010/0/1019/1025/1049	1.12692/1.31406
51	AB9721	A	M	1030	1035/0/1022/1030/1052	1.97607/1.17358
5	AB6300	D	M	1030	1015/0/1023/1030/1053	1.44608/1.03506
30	CL2301	A	M	1032	1037/0/1029/1032/1059	1.62744/1.01266
8	A3501	D	M	1034	1020/0/1028/1035/1058	1.20044/1.36356
41	CL2175	A	M	1035	1040/0/1031/1035/1101	1.44662/1.50711
7	AY804	D	M	1035	1020/0/1032/1035/1102	1.2627/1.01184
52	CL2491	A	M	1039	1045/0/1030/1040/1100	1.08829/1.73813
2	LH107	D	M	1040	1000/1040/1035/1040/1050	1.25311/1.75067
55	U23411	A	M	1044	1050/0/1043/1045/1113	1.04108/1.68204
10	TK1630	D	M	1045	1030/0/1043/1045/1113	1.69086/1.98947
11	LY354	D	M	1046	1030/0/1037/1045/1107	1.18015/1.43527
13	AB8282	D	M	1054	1035/1055/1050/1055/1105	1.15308/1.56119
59	LH2033	A	M	1055	1100/0/1045/1055/1115	1.85117/1.36209
15	LH2066	D	M	1055	1040/0/1048/1055/1118	1.73782/1.30496
19	CL2442	D	M	1059	1045/0/1057/1100/1127	1.25046/1.94989
16	EN1950	D	M	1100	1045/0/1055/1100/1125	1.90781/1.79709
18	CL2272	D	M	1101	1045/0/1052/1100/1122	1.93961/1.0094
21	LH2286	D	M	1115	1045/1115/1110/1115/1125	1.38727/1.20053

Appendix B: Abstract

This master thesis deals with the single and multiple runway cases of the Aircraft Sequencing Problem involving departing and arriving airplanes. In detail, it deals with the problem of assigning each departing and arriving airplane to an appropriate runway, computing a sequence for each runway, and scheduling take-off and landing times for each airplane. Thereby, each flight has to depart or land within an individually specified time window. In addition, it must be ensured that minimum separation requirements between airplanes are strictly observed.

Following a number of authors, in this thesis, the underlying problem is formulated as static decision problem, exclusively considering immediate runway traffic. The linear objective function adopted corresponds to the minimization of total weighted deviation from individual target times. In this context, target times are defined as preferred take-off or landing times, i.e. points in time, where no additional airplane-specific costs incur.

As the problem in this work is considered as routing problem, a Variable Neighborhood Search metaheuristic, basically designed for problems of transportation logistics, is presented as solution method.

The performance of the implemented heuristic is measured using 13 test sets provided by the OR-library of J.E. Beasley. Computational results for all 49 test instances, involving from 10 to 500 airplanes, in the presence of single and multiple runways are presented. In addition, a comparative analysis with current “state-of-the-art” heuristics is shown. The implemented algorithm provides excellent results, outperforming all but one heuristics previously presented. Nevertheless, drawbacks regarding runtimes exist, especially as problem size increases.

Finally, an application of the algorithm to a real world scenario encountered at Munich International Airport is presented. A comparative analysis with a “first-come first-serve”-methodology, as is still prevailing at most large international airports, reveals potential cost savings up to approximately 70%. This emphasizes the effectiveness of optimized runway operations management.

Appendix C: Zusammenfassung

Die vorliegende Diplomarbeit behandelt das „Aircraft Sequencing Problem“ für den Fall einer oder mehrerer zur Verfügung stehender Start- bzw. Landebahnen, unter Einbeziehung von startenden als auch landenden Flugzeugen. Das bedeutet, es werden abfliegende und anfliegende Flugzeuge sowohl zu Pisten zugeteilt, als auch die Reihenfolge derselben auf jeder einzelnen Piste bestimmt. Weiters ist es notwendig, Start- und Landezeiten eines jeden Flugzeuges festzulegen. Dabei müssen für jeden Flug individuell zugewiesene Zeitfenster berücksichtigt werden. Zusätzlich muß sichergestellt sein, daß die erforderlichen Mindestabstände zwischen den Flugzeugen eingehalten werden.

Einer Reihe von Autoren folgend, wird das zugrundeliegende Problem als statisches Entscheidungsproblem, ausschließlich den Flugverkehr im unmittelbaren Pistenbereich berücksichtigend, betrachtet. Als lineare Zielfunktion dient die Minimierung der Summe der gewichteten Abweichungen von individuellen Zielzeiten. Das sind jene präferierten Start- bzw. Landezeiten, die keine zusätzlichen flugzeugspezifischen Kosten verursachen.

Da die zugrundeliegende Arbeit das Problem als Transportproblem betrachtet, wird eine „Variable Neighborhood Search“-Metaheuristik, welche ursprünglich für Problemstellungen der Transportlogistik entworfen wurde, als Lösungsverfahren vorgestellt.

Die Leistungsfähigkeit der implementierten Metaheuristik wird anhand von 13 Testdatensätzen der „Operations Research“-Bibliothek von J.E. Beasley gemessen. Ergebnisse von 49 Instanzen, 10 bis 500 Flugzeuge umfassend, werden im Ein- und Mehrpistenfall präsentiert. Zusätzlich wird eine Vergleichsanalyse mit den derzeit besten Heuristiken dargestellt. Das implementierte Verfahren liefert exzellente Resultate, die die Ergebnisse beinahe aller bislang bekannten Heuristiken übertreffen. Nichtsdestotrotz weist es Nachteile hinsichtlich der Laufzeiten auf, speziell bei zunehmender Problemdimension.

Abschließend wird die Anwendung auf eine reale Problemstellung des Flughafen München erläutert. Eine Vergleichsanalyse mit einem „first-come first-serve“-Ansatz, der nach wie vor auf den meisten internationalen Flughäfen gängige Praxis darstellt, bringt potentielle Kosteneinsparungen von bis zu 70 % zum Vorschein. Das unterstreicht die Effektivität von optimiertem operationellen Flugverkehrsmanagement.

Appendix D: Curriculum Vitae

Personal Data

Name: Thomas Hermann
Date of Birth: 19.11.1975 in Vienna, Austria
E-Mail: thomas.hermann@austrian.com

Professional Experience

Since 2004: Austrian Airlines AG, Vienna, Austria
Airline Transport Pilot
First Officer Boeing 737

1999 – 2004: Lauda Air Luftfahrt AG, Vienna, Austria
Airline Transport Pilot
First Officer Boeing 737

08/1998: Lauda Air Luftfahr AG, Vienna, Austria
Trainee in Department for Quality Assurance

1992 – 1994: Bundesamt für Zivilluftfahrt, Vienna, Austria
Trainee in Technical Department of Air Traffic Control during summer terms

Education

Since 1996: University of Vienna, Vienna, Austria
Master Program in International Business Administration
Core subjects: *Production Management, Financial Engineering*
Master Thesis: "Variable Neighborhood Search for the Static Aircraft Sequencing Problem"

1990 – 1995: Secondary Technical College, Vienna, Austria
Telecommunication Engineering
Thesis: “Microprocessor-based Frequency Generator”
Degree: A-levels

1986 – 1990: Secondary School, Vienna, Austria

1982 – 1986: Primary School, Vienna, Austria

Language Skills

German: mother language
English: fluent in written and spoken
Spanish: good in written and spoken

Computer Skills

Software: MS-Office, SAP (MM-module), R
Programming: C++, Visual Basic, Modula 2