Erasmus University Rotterdam

Tinbergen Institute / Business Data Science

Teachers: Bernd Heidergott, Ad Ridder

Aircraft Boarding Strategies Using Discrete Event Simulation

Abstract

As the need for optimal boarding strategies which can substantially reduce the amount of delays is becoming more important, more research studies have aimed to investigate the optimal boarding strategy by means of discrete event simulation. The objective of this report is to find a boarding strategy that minimizes boarding time by simulating aircraft boarding strategies through discrete event simulation, where the movements for each passenger are modelled by uniform distributions and the arrival of each passenger to the plane is modelled through an exponential distribution. We find that the window-middle-aisle boarding strategy is found to be the most efficient, followed by the optimal boarding strategy, in terms of total boarding time, the CVAR 95% value and the mean individual boarding time.

Keywords: JEL C15, C63, discrete event simulation (DES)

Author:	*Student ID:*
David Gyaraki	582340
Lavin Pallas	581619
Julius de Clercq	582360

April 2, 2023

Contents

1 Introduction

The purpose of this report is the simulation of an aircraft boarding system and the evaluation of the system using different boarding strategies based on the paper of Jafer & Mi (2017) with the use of discrete event simulation. It involves events occurring at specific points in time, and the system state changing as a result of each event. In such simulations, time is discrete, which each point in time corresponding to the occurrence of an event. Hence, the system state is updated only at these discrete points in time. Generally, the events in discrete event simulations can be the start or end of a process in general, or any other event that can be modeled as discrete (either by nature or by simplifying the system into a discrete event using assumptions) and occurring at a specific time. In our simulation, these events include the arrival of a passenger to the plane, the movement of a passenger in the plane, and the moment a spot on the aisle is cleared due to a passenger having been seated.

Literature suggests that the most efficient boarding strategies consist of the Window-Middle-Aisle and its variant, the Reverse Pyramid boarding strategy whereas some of the less efficient strategies have been found to be the Back-to-Front strategy (Jafer & Mi, 2017). It has been found that as congestion increases, the order among boarding groups becomes less relevant and the various sizes of the groups are the only parameters that count. The total boarding time for the random boarding strategy is computed using the Euler-Lagrange equation for different time step values k as the joint distribution of the queue is uniform (Bachmat et al., 2009a). Nevertheless, according to Bachmat et al. (2009b), the dynamically optimized boarding strategy has been found to be more efficient compared to the window middle aisle seat strategy, which involves both allowing passengers to board in cliques, such as families with young children for instance, as well as depending on their seat. The most essential parameters considered for simulation consist of the passenger's walking speed, the luggage storage time, the time to clear a seat interference, and the passenger inter-arrival time. The seat interference potentially involves 0, 1 or 2 seats depending on the number of passengers (and the specific aircraft models), and this applies for this simulation study as well.

According to Ferrari & Nagel (2005), the half block boarding strategy consisting of only one group was the most efficient, as well as alternating rows which eventually was found to be too complicated however. The most inefficient strategy was proven to be the back to front boarding strategy as well as block boarding with many sub passenger groups. A deterministic process whereby each passenger can move one cell or row forward per time step has been implemented. In addition, a different number of disturbances was introduced consisting of early and late passengers, the aircraft occupancy as well as the aircraft dimensions, which play a role in the quality of the boarding strategy. The seating model which determines the boarding time is determined by the sum of the total time for seating plus two times the time to get from seat into aisle and back multiplied by the number of obstructing passengers who are already seated.

Another research study by Van Landeghem & Beuselinck (2002) who were among the first scholars to study aircraft boarding strategies, found that calling all passengers individually by their row and seat number is the fastest approach, yet the riskiest and most complicated method if anything went to go wrong, which can lead to confusion and a tremendous amount of delays. A more realistic approach is random boarding which leads to the overall reduction of total boarding time, nevertheless it results individual boarding time delays. The boarding time was calculated by taking the sum of the seating time of

all passengers including the delays of each individual, the amount of time necessary to store luggage, delays within the same class caused by individuals attaining their seat and delay among different classes caused by interference between consecutive classes. In this study a triangular distribution for passenger movement was assumed.

The rest of this paper follows as: Section 2 describes the conceptual model of our system used in the simulations as well as the events and different boarding strategies used; Section 3 briefly describes the programs, functions and random distributions used for the simulation; Section 4 touches upon the use of parameters and their practical viability compared to other papers; Section 5 summarizes the simulation results using the evaluation metrics; Section 6 elaborates on the results and discusses the optimal boarding strategies in light of our result and finally, Section 7 concludes.

2 Conceptual model

This section will first briefly describe the conceptual model we use for Cellular DES of an aircraft boarding system, and then elaborate a bit more on the system states, events and transformations that can occur within our system, and then describe the different boarding strategies used for testing this system. To develop the conceptual model of our system, we made a number of different assumptions, sometimes following Jafer & Mi (2017) and in other cases making our own assumptions of the system. These assumptions help us construct a concept of an aircraft and the boarding of this aircraft which we can then test.

For aircraft type, we picked an Airbus 320 with the arrangement of seats applied by WizzAir [1]. Therefore we assume an aircraft with symmetric sides, 30 rows of equal length seats (no business class), 3 seats on each side of the aisle. The draft of the model we follow can be found also in the Appendix in the boarding strategy visualization graphs (such as Figure 3). The equal seat length is an assumption which does not hold in reality since emergency exit rows have to be longer, but we can simplify this to avoid confusion over row and aisle cell length heterogeneity. The general cell measure in our system is one seat, which is 28 inches long and 18 inches wide [1], which corresponds to 71.12cm cell length and 45.72cm cell width. Similarly to Jafer & Mi (2017), we assume that the aisle space in a row corresponds to a single seat cell equivalent, i.e. the aisle is 71.12cm long between two seats and 45.72cm wide. In addition to the aisle among the rows, we assume that the entrance to the rows of seat consists of one more aisle cell and 4 aisle cells which the passengers have to traverse on a cross-way direction. This means that the passenger (from entering the aircraft) needs to cover $4 \times 45.72cm$ and once 71.12cm to reach the first row.

Furthermore, we assume that the airline can only conduct front row boarding in order to reduce complexity of front-rear boarding and questions arising from this, such as the problem of non-complying passengers, who board in the front even though they are assigned to rear and vice-versa. This means that the aircraft can only be boarded from the front, for example the airplane is directly connected to the airport terminal. Furthermore, as Figure 3 also shows, apart from the aisle cells and the seats, all other space is assumed to be inaccessible for the passengers, i.e. they cannot walk towards the cockpit, they cannot walk past the last row, etc. One cell may only contain one passenger at any given time with one exception which will be mentioned later. The aisle cell is said

[1] The aircraft seat layout and parameters can be found at SeatGuru.

to contain the passenger from the time the passenger passes the boundary of the aisle and the cell is freed up again when the passenger crosses the other boundary. Therefore we assume that passengers may not pass each other on the aisle, so if a passenger stops on a certain aisle cell, another second passenger behind needs to stop on the previous aisle cell in case they catch up and wait until the first passenger progresses or sits down.

Furthermore, we assume that from the moment a passenger enters the aircraft, they progress in a single direction on the aisle (no back-tracking) and goes to their row (walking continuously or stopping to wait for another passenger). Passengers are all assumed to know their seat exactly and cannot choose a different seat. Once the passenger reaches their row, they stop and first stash their luggage (in the overhead compartment or under the seat) if they have any. Since a passenger can stash the luggage both under the seat and overhead, we assume that the airline made sure before the boarding that there is enough space for all passengers' luggage where they are sitting (or within reach) and there is no need for the passengers to move back and forth on the aisle to find space for their luggage. After the luggage (if the passenger has any) is stowed away, then begins the seating procedure for that passenger (i.e. the passenger has to stow their luggage before sitting down). However, it is possible that a passenger arrives to their row to find that someone is already sitting there, blocking their way to the seat.

In this case, make the following assumptions for the system. In order for any passenger to reach their seat, first any potential passenger blocking the access needs to stand up (the arriving passenger cannot squeeze past the already sitting passengers). For example, if a passenger has the window seat, but the row they are sitting on already has the middle and aisle seat occupied, then first the aisle seat passenger needs to stand up, then the middle seat passenger needs to stand up, then the passenger who arrived can take their seat, then the middle seat passenger can sit back and finally the aisle seat passenger can take their place again. In the case of standing up however, we assume that the aisle cell on the same row can host multiple people, by the one or two people standing up also stand in this cell, or leaning towards the other side of the row, etc. so the stand-up does not take up the preceding or succeeding aisle cell. While this may not be completely realistic (to fit two or potentially even three passengers in the same row and aisle space standing), we can potentially make use of this simplifying assumption to avoid unsolvable blockage and back-tracking situations, for example if two people need to stand up on the last row with no runoff area to go to and the aisle space preceding the last row is blocked as well. Furthermore, the aisle space is assumed to be blocked until either the arriving passenger has taken their place (if nobody had to stand up) or until the last passenger who had to stand up takes their place once again. After having made these simplifying assumptions about our system, we can move on to describing (some of) the events, states and transitions our system makes use of when simulating the boarding process.

2.1 Events and states

Table 1: List of events and states

events	**states**
α - passenger entering the door of aircraft	A_k - aisle space cell k is occupied [yes(1) or no(0)]
β - passenger walks one aisle cell	P_{ij} - passenger of row i and seat j in the plane, unseated [yes(1) or no(0)]
γ - passenger reaches their row	S_{ij} - seat j in row i occupied by passenger [yes(1) or no(0)]
δ - passenger stops without reaching their row	
ω - passenger takes their own place	

Our state space can be specified by the following arrays of individual state values as described in Table 1:

$$A = \begin{bmatrix} \{0,1\} & \{0,1\} & \{0,1\} & \{0,1\} \\ & & & \{0,1\} \end{bmatrix} \quad P,S = \begin{bmatrix} & & & & & \\ & & & & & \\ \{0,1\} & \{0,1\} & \{0,1\} & \{0,1\} & \{0,1\} & \{0,1\} \end{bmatrix} \tag{1}$$

where we have a restriction on the combined values of S and P in such a way that

$P_{ij} + S_{ij} \leq 1 \; \forall i \in \{1,\ldots,30\}, j \in \{1,\ldots,6\}$, which takes care of the restriction that a passenger cannot be in the plane walking (or waiting) and seated at the same time. Due to the large number of cells, each with their own state, it would be a massively complex list to describe all possible events in each state, but we can write up the transitions on the state space with events happening for one passenger. For passenger i,j, the events in certain states (Table 1) can occur in the following ways:

$$
\begin{aligned}
&L(A_1 = 1, P_{ij} = 0, S_{ij} = 0) = \emptyset \\
&L(A_1 = 0, P_{ij} = 0, S_{ij} = 0) = \{\alpha\} \\
&\text{If } P_{ij} = 1, \text{ then let the aisle cell taken up by } P_{ij} \text{ be denoted by } A_k, \text{ where } k-5 \leq i \\
&L(A_{k+1} = 0, 1, 0) = \{\beta\} \; if \; k-5 < i \\
&L(A_{k+1} = 1, 1, 0) = \{\delta\} \\
&L(A, 1, 0) = \{\gamma\} \; if \; k-5 = i \\
&L(A, 1, 0) = \{\omega\} \\
&L(A, 0, 1) = \emptyset
\end{aligned}
\tag{2}
$$

Alternatively, we can line up the transitions on the state space each event occurrence causes in the following ways:

$$
\begin{aligned}
&\phi\Big(([0, A_2, \ldots, A_{35}], 0, 0), \alpha\Big) = ([1, A_2, \ldots, A_{35}], 1, 0) \\
&\phi\Big(([\ldots, A_{k-1}, 1, 0, \ldots], 1, 0), \beta\Big) = ([\ldots, A_{k-1}, 0, 1, \ldots], 1, 0) \\
&\phi\Big(([\ldots, A_{k-1}, 1, 1, \ldots], 1, 0), \delta\Big) = \text{no change on state space} \\
&\phi\Big(([\ldots, A_{k-1}, 1, \ldots], 1, 0), \gamma\Big) = \text{no change on state space} \\
&\phi\Big(([\ldots, A_{k-1}, 1, \ldots], 1, 0), \omega\Big) = ([\ldots, A_{k-1}, 0, \ldots], 0, 1)
\end{aligned}
\tag{3}
$$

2.2 Boarding strategies

Our conceptual boarding model provides the system we use for simulating an aircraft boarding procedure, but it also raises the question of evaluation and optimisation of the system, i.e. how do we measure efficiency and how to make the functioning of the system more efficient. As Jafer & Mi (2017) describes a number of different boarding strategies, most of which with foundations in real life cases, our paper will utilize these boarding strategies as tools to evaluate the system. The list of these boarding strategies are: Back-to-Front (BF), Window-Middle-Aisle (WMA), Rotating Zone (RZ), Optimal (O), Practical-Optimal (PO), Reverse Pyramid (RP) and Random (R). The most obvious evaluation metric of these strategies applied on our boarding system is the total boarding time. From an airline's perspective, perhaps the most important measure of different boarding strategies is the time it takes to receive and host all passengers, from the time boarding is started until all passengers stow their luggage and take their seats. Another important metric could be the individual time a passenger spends boarding the aircraft

with specific attention to not only the average but the values in the tails. The assumption here is that passengers generally do not enjoy standing, queuing and changing seats while within the narrow and crowded space of an aircraft. In the following, we will describe the boarding strategies also presented by Jafer & Mi (2017) in detail and in particular where we deviate from their methodology. One of the major deviations is that our aircraft model assumes no business seats (seats in a particular area) and no priority boarding, i.e. all seats are treated as equal priority for the boarding strategies.

The *Back-to-Front* (BF) strategy means that the 30 row aircraft is divided into 6 boarding groups and their corresponding zones, where each zone is called upon to board one after the other. Like in all the other cases where we have some systematic strategy, we make a simplifying assumption that passengers are compliant with the boarding policies, that is they all board when they are called upon (such as having a boarding group designation on their boarding pass). The BF strategy designates zone 1 as rows 1-5, zone 2 as rows 6-10, zone 3 as rows 11-15, zone 4 as rows 16-20, zone 5 as rows 21-25 and zone 6 as rows 26-30. Then we begin the boarding from the back, i.e. passengers arrive randomly from zone 6 and once all of them arrived, we begin calling passengers from zone 5 and so on until everyone boards from zone 1. The order of arrival within the zone groups is sampled randomly without replacement. Intuitively, this strategy might be very helpful to avoid congestion along the aisle (people seated in the front will not keep back people in the back for long), but may be costly in terms of having to resolve passengers hopping in and out of their seats to let each other pass to their seats.

Similarly, the *Window-Middle-Aisle* (WMA) strategy creates 3 zones and calls upon the passengers there, who arrive randomly within the zones. Zone 1 contains all the window seats on both sides of the airplane, zone 2 contains all middle seats and zone 3 contains all aisle seats in all rows on both sides. This strategy may be very helpful in resolving issues presented by the BF strategy, namely eliminating the need for people to stand up and sit down again to let each other pass to their seats, but complementarily may cause long queues and blockage in the aisle.

Then the *Rotating Zone* (RZ) strategy addresses the issue of this blockage by changing the order of the boarding zones in the following fashion: zone 6 (rows 26-30) boards first so that the back zone can be eliminated. Then zone 1 (rows 1-5) is boarding to eliminate potential interference with the ones finishing the boarding in the back. Then zone 5, zone 2, zone 4 and zone 3 in this order. The rotation of the zones is meant to attempt to eliminate the interference among the groups.

The *Reverse Pyramid* (RP) strategy attempts to combine the BF and WMA strategies and harvest the benefits of both at the same time by building a 6 symmetric zone (both sides of the aircraft are used at the same time) boarding order. In our methodology, the groups are defined in a slightly different logic than the one described by Jafer & Mi (2017). Zone 1 consists of the window seats on rows 16-30. Zone 2 contains the window seats of rows 8-15 and middle seats of rows 24-30. Then zone 3 contains the window seats of rows 1-7 and middle seats of 16-23. Zone 4 contains the middle seats of rows 8-15 and aisle seats of 24-30. Zone 5 contains the middle seats of rows 1-7 and aisle seats of 16-23. And finally, zone 6 entails the aisle seats of rows 1-15. The boarding order is also the same, i.e. zone 1 starts and zone 6 finishes the boarding.

Then the *Optimal* (O) strategy details a very detailed zoning technique similar to Jafer & Mi (2017) where we define 12 zones based on the row number being odd or even, the position of the seat (WMA) and the side which they are on. Zone 1 consists of all window seats on even rows on one side (left) of the plane, zone 2 includes all window

seats on even rows on the opposite (right) side, zone 3 entails the window seats on the odd rows on the left side of the plane, zone 4 contains all window seats on odd rows on the other (right) side of the plane, and then same is repeated for middle seats in zones 5-8 and for aisle seats in zones 9-12. Boarding order among zone groups is sequential from 1 to 12. The advantage of this strategy is that it combines the WMA strategy with the BF in a way that it tries to avoid congestion resulting from passengers trying to board the same zone in the BF strategy. However, questions clearly arise in terms of the viability of practical implementation of such a strategy due to the underlying complexity.

In addition to the optimal, the *Practical-Optimal* (PO) strategy attempts to simplify the strategy by only considering the odd and even rows. In this strategy, zone 1 consists of all even row seats (with no WMA distinction) on the left side of the aircraft, zone 2 contains all even rows on the right side, zone 3 entails all odd row seats on the left and zone 4 the odd row seats on the right side of the plane. Similarly to the previous strategy, the boarding order of the zone groups is sequential, so it progresses from 1 to 4.

Finally, the *Random* (R) strategy simply randomizes the order of the arriving passengers regardless of their seating positions. This boarding strategy may serve as the default case of the boarding if the airline does not want to pay attention or put effort into defining boarding groups. Alternatively, it may also serve as a counterfactual to the strategies in case of a high rate of non-compliance, i.e. if passengers disregard (intentionally or accidentally) the calls for their own boarding groups and decide to enter randomly. Albeit the assignment of passengers to certain groups (always takers, compliers, etc.) may not be random to their characteristics, we may assume that their assigned seats are random to both their individual characteristics and their compliance groups.

3 Simulation program

Our simulation procedure consists of a number of functions managing the aisle space of the aircraft, the waiting delays and the seating manager taking care of luggage and shuffling of passengers on the same row. In this section, we will present the main logic of these functions along with the parameters and distributions used and the flowchart of the logic and events. First of all, we can discuss the attributes of the passengers boarding the plane and the random variables pertaining to the passengers. The passengers are assumed to have different walking speeds, which we estimate by a uniformly distributed random variable s. This variable describes the passengers' speed in meters per second, which they walk the aisle (we assume continuous and even walking speed, no acceleration or deceleration). Then we estimate the entrance time of each arriving passenger by an exponentially distributed random variable e, which is described in Table 2.

Table 2: List of random variables used for passenger attributes and seating management

Parameter description	**Distribution**
Walking speed of the passenger (m/s)	$s \sim U(0.27, 0.44)$
Seconds until passenger entrance/arrival	$e \sim Exp(\lambda = 1/2)$
Seconds it takes to stow away luggage	$l \sim U(6, 30)$
Seconds it takes to sit down/stand up aisle seat	$a \sim U(2, 5)$
Seconds it takes to sit down/stand up middle seat	$m \sim U(3, 6)$
Seconds it takes to sit down window seat	$w \sim U(4, 7)$

Figure 1 presents a flow chart describing the discrete event simulation that dictates the arrival of passengers aboard the plane and their subsequent movement on the aisle towards the row at which they have their seat. The motor of the program is the global time, which for each iteration of the simulation is updated to the time that the next event occurs. The event that occurs can either be the arrival of a new passenger aboard the plane, or an action of a passenger that has already entered the plane.

If the event concerns the arrival of a new passenger, the program first evaluates whether the first spot on the aisle is vacant or occupied. If occupied, the new passenger cannot enter, and it's arrival is delayed by one second. This admittedly is an arbitrary number; the reason for it being that neither a new draw from the usual exponential distribution for the arrival of a new passenger applies, as this passenger is already waiting at the door, nor the usual time to move to the next spot of the aisle applies, as crossing the threshold concerns a different distance. Therefore we do not have enough information to give a well-substantiated input, and we set this time to a seemingly reasonable update of one second. If the first spot on the aisle is vacant, the passenger enters the plane and now occupies that first spot. The next action of this passenger necessarily is movement to the next spot on the aisle, as the rows with seats do not appear until some way up the aisle (five tiles in our case). Hence, its event (i.e. action) timer is updated to be the time it takes for this passenger to move to the next spot, which is the length of the spot on the aisle (which is slightly longer for the tiles that are adjacent to seats, as at those spots the passengers tend to occupy more space) divided by the passenger's walking speed.

If the event does not concern a new arrival, we are necessarily dealing with the action of a passenger that has already entered the plane. If this passenger has already been seated, it means that the spot on the aisle is now vacant, as the event that has just occurred implies that all actions that cause the spot on the aisle to be occupied (i.e. stowing luggage, sitting, and seated passengers standing up to make room) are now completed. As this passenger is now seated, we remove her actions from the event list. This implies that once the last passenger is seated, there are no more scheduled events and the simulation is finished.

If the passenger has not yet been seated, it implies that the row at which the passenger is seated has not yet been reached, meaning that the passenger needs to move up the aisle. If the next spot on the aisle is occupied, the event timer of the passenger is updated to the time it takes to move from the current spot to the next spot on the aisle, and this time is added to this passenger's waiting time. If the next spot on the aisle is vacant, the passenger moves to the next spot on the aisle and both the state of the aisle (i.e. aisle occupancy or vacancy) is updated, and the state concerning the passenger's position on the aisle is updated. If this new position does not yet correspond to the passenger's

seating row, we simply schedule the passenger's next movement (i.e. event) to be the time it takes to reach the next spot on the aisle.

But if the passenger has indeed reached its seating row, the seating procedure is initiated, which will be described in detail in the next paragraph. For the purposes of the event simulation, we just need to know that this procedure dictates the time that it takes for the aisle to be cleared, which accounts for stowing luggage, sitting and seated passengers standing up to make room for the arriving passenger. The next event for this passenger will be the moment that the passenger is seated and the aisle is no longer occupied. Therefore we already mark this passenger as seated and set the timer of its next event to the time it takes for the aisle to clear.

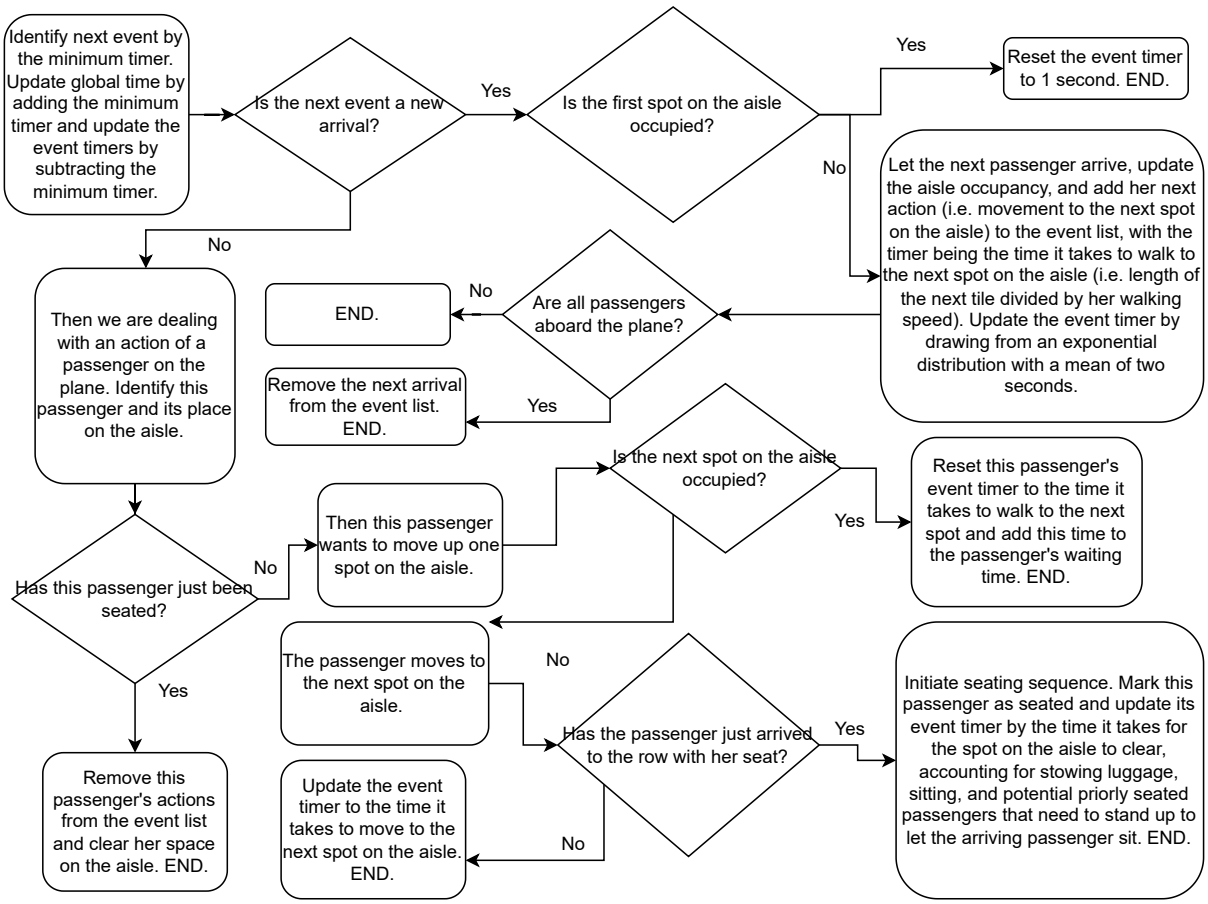

Figure 1: Flow chart of the discrete event simulation, concerning passengers arriving on the plane and moving on the aisle. This is simulation is repeated until all passengers are seated, implying the event list to be empty.

Then once the passenger reaches their own row, they stop on the aisle cell and the seating management process begins. Figure 2 presents the flow of logic once the passenger arrives to their row. First, the passenger takes care of stowing away their luggage if they have any, which takes a random amount of time estimated by a random parameter l which follows a uniform distribution (Table 2). Then after the potential luggage stash, we need to establish if the arriving passenger has a window, middle or aisle seat. If the passenger has the aisle seat ($j = \{3, 4\}$), they can take their seat regardless of whether other passengers are sitting on the same row already. The time to take the seat and vacate the aisle cell takes a random parameter a from a uniform distribution. If the arriving

passenger has a middle seat ($j = \{2,5\}$), we need to investigate whether the aisle seat is occupied already. If not, then the arriving passenger takes the middle seat using up time estimated by a parameter m from a uniform distribution. If the aisle seat is taken, then the passenger there needs to stand up (a), then the arriving passenger takes their seat (m) and then the aisle passenger sits back, vacating the space (time spent on this is again the same a drawn from the distribution earlier).

Finally if the arriving passenger has a window seat ($j = \{1,6\}$), we have 4 cases. First is if both the aisle and middle seats are free, in which case the passenger takes their seat with a time parameter w from another uniform distribution (Table 2). The second and third cases entail when either the aisle or the middle seat is occupied but the other one is not. In this case, the occupying passenger stands up (a or m depending on seat), the arriving passenger takes their place (w) and then the other passenger sits back, vacating the aisle cell. Fourth and last is if both aisle and middle seats are occupied. In this case, first the aisle seat passenger stands up (a), then the middle passenger stands up (m), then the arriving passenger takes their seat (w), after which the occupying passengers sit back in the same time variables.

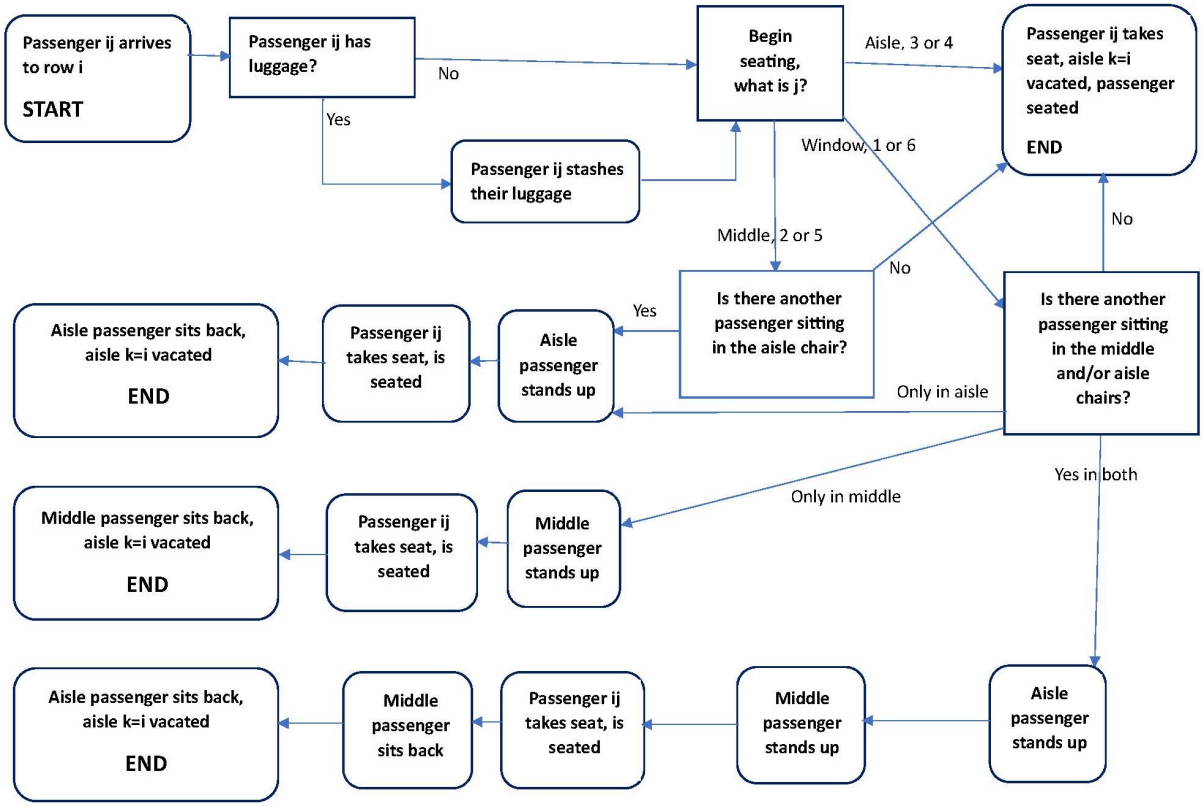

Figure 2: Flow chart of the seating procedure

4 Validation and Verification

Research by Chun & Wai Tak Mak (1999) suggests the use of an exponential distribution for passenger arrival and check in processing time, with a time varying mean parameter. The average waiting time resulting from the simulation is found to be 10 minutes when the counter profile is equipped with sufficient staff. The maximum waiting time is found to be 55 minutes, the maximum queue length is found to be 8 minutes whereas the average

queue length is 1.4 minutes. Another research study Schultz (2008) used the triangular distribution to model the storage process which was found to be 13.9 seconds as well as the Weilbull distribution with the scale parameter $\alpha = 1.7$ and shape parameter $\beta = 16$ seconds. The study by Van Landeghem & Beuselinck (2002) suggested γ is the number of capacity of luggage space which was found to be 6, α which is defined as the time to stow luggage was found to be 2 and β is the correction coefficient which was found to be 11. According Qiang et al. (2018) who used the logit model to model seat choice behavior the parameter V which refers to the attraction value of seat was estimated to take a value of 9.1. In addition , the suggested optimal value for the scaled parameter θ which measures preference difference toward the two successive takes the value of 0.15 and leads to a delay of the boarding time when the value increases. The parameter ϵ which measures the exclusion effect between individual passengers in the same half row is found to be 0.5, as the value of the parameter increases, it will lead to passengers select seats in different rows. Finally the simulation results by Jafer & Mi (2017) who found that the window middle aisle boarding strategy which was the fastest took 26 minutes suggest the parameters walking speed is between 0.27 and 0.44 m/s, the clearing time is between 6 and 30 seconds, the getting out of a seat is between 3 and 4.2 seconds and the passenger flow rate is between 0.2 and 1 pax/s. According to the random variables for seating management which have been generated from a uniform distribution, the time to stow luggage takes 6 seconds on average with a variance of 30 seconds which is lower than the results of Schultz (2008) who found the time to stow luggage to be approximately 13.9 seconds. The number of seconds to sit down and stand up from an aisle seat has been estimated to be on average 2 seconds with a variance of 5 seconds. Similarly, The number of seconds to sit down and stand up from a middle seat has been estimated to be on average 3 seconds with a variance of 6 seconds and from a window seat 4 seconds on average with a variance of 7 seconds. The findings of Jafer & Mi (2017) indicate that getting in and of a seat takes approximately between 3 and 4.2 seconds depending on the seat, which indicate that the results from this study are quite in line with their results. The walking speed was estimated to be approximately 0.2 seconds on average with a variance of 0.44 whereas the findings of Jafer & Mi (2017) also indicate that the walking speed was estimated as approximately between 0.27 and 0.44 miles per second which is quite in line with the results of this study.

5 Results

In our evaluation of the different boarding strategies, we make a simulated list of passengers corresponding to a particular boarding strategy, repeat this 100 times for each strategy and run the simulation. Then we evaluate each strategy by the average individual boarding time, the 95th percentile c-var measure of the same individual boarding time and the total boarding time from start to finish.

Table 3: Results of a 100 simulations for each boarding strategy, with the average time in minutes across the 100 simulations for each strategy

Boarding strategy times	Mean individual boarding	CVAR95 individual boarding	Total boarding
Back-to-front (BF)	4.025	6.604	38.821
Window-Middle-Aisle (WMA)	1.968	3.141	24.451
Rotating-Zone (RZ)	3.483	5.929	35.941
Optimal (O)	1.901	3.024	23.835
Practical-Optimal (PO)	2.208	3.566	27.617
Reverse-Pyramid (RP)	2.256	3.862	26.325
Random (R)	2.259	3.695	28.071

Table 3 provides us with the average times across the 100 simulations with regards to the mean passenger boarding time and cvar (mean in the right tail) as well as the total boarding time from start to finish (first to last passenger). The back to front strategy has a mean individual boarding time of approximately 4.025 seconds, a CVAR95% of approximately 6.604 seconds and a total boarding time of 38.821 seconds. The window middle aisle strategy has a mean individual boarding time of approximately 1.968 seconds, a CVAR95% of approximately 3.141 seconds and a total boarding time of 24.451 seconds. The rorating zone strategy has a mean individual boarding time of approximately 3.483 seconds, a CVAR95% of approximately 5.929 seconds and a total boarding time of 35.941 seconds. The optimal strategy has a mean individual boarding time of approximately 1.091 seconds, a CVAR95% of approximately 3.024 seconds and a total boarding time of 23.835 seconds. The practical optimal strategy has a mean individual boarding time of approximately 2.208 seconds, a CVAR95% of approximately 3.566 seconds and a total boarding time of 27.617 seconds. The reverse pyramid strategy has a mean individual boarding time of approximately 2.256 seconds, a CVAR95% of approximately 3.862 seconds and a total boarding time of 26.325 seconds. Finally, the random strategy has a mean individual boarding time of approximately 2.259 seconds, a CVAR95% of approximately 3.695 seconds and a total boarding time of 28.071 seconds.

6 Discussion

Having conducted the simulations and having the results, we can briefly elaborate on the results in the wider context and address the simplifications in light of our results. Table 3 showed us that the best strategies are the WMA and Optimal strategies in terms of all 3 measures. On the other end of the spectrum, BF and RZ strategies perform visibly worse in all measures. One reason for such a difference across these strategies could be the row congestion. BF and RZ strategies tend to sample from zones which have seats very close to one another. Even though the idea is that these zones avoid interference from one another, it seems that passengers within each boarding group hold each other up significantly.

On the other hand, 3 strategies perform relatively similarly, the PO, RP and Random strategies. This similar performance on our boarding system brings up the question of practicality. The PO strategy is relatively simple, only uses the different sides of the aircraft and odd-even rows, but does not specify the order of seats (W-M-A) in a row. On the other hand, the Reverse-Pyramid strategy is a rather complicated strategy trying to fuse the seat and row orders into one strategy using 6 zones, if these strategies are not better than the Random strategy, which requires no sophistication and coordination of passengers prior to boarding, the additional resources spent on organising and supervising these boarding strategies makes them obsolete. Therefore, one should consider the Random strategy as a benchmark to beat for a specific boarding strategy to justify the effort. Specifically, while the mean total boarding time across the 100 simulations is larger for the random strategy than the RP by about 1.75 minutes, the longest waiting passengers experience higher average waiting times than in the random case.

Hence we should focus on the Window-Middle-Aisle and the Optimal strategies, since they reduce the times from a Random boarding order, in terms of total boarding time by 3.5-4 minutes respectively and the average passenger boarding time by 18-21 seconds. This seems like a good way to improve boarding efficiency, potentially raise airline customer satisfaction and avoid large boarding delays which disrupt the airline's flight schedules and may incur extra costs. For example if a WizzAir Airbus 320 has nine 2 hour long short-haul flights planned for a day, the 8 turn-arounds can save around 30 minutes per day. Comparing the two strategies against one another also gives us some indications on which one should be used. While the optimal strategy performs a little bit better, the results of the WMA are almost as good. However, the Optimal strategy uses a highly complex, 12-zone boarding strategy, which is rather difficult to organise, coordinate and enforce. On the other hand, the WMA strategy very simply uses the 3-zone strategy, with window seats boarding first, then middle and aisle. This makes the use of this strategy significantly more realistic and advisable.

However, these results are valid within our system of assumptions, which also show the limitations of the system and the simulations to the real life cases. First of all, since we assumed the aircraft to be a WizzAir Airbus 320, there are a few assumptions which do not hold in real life. For example, WizzAir often makes use of double-door (both front and back simultaneously) boarding systems if it is possible in the particular airport it services. Then often the airline offers priority boarding, which would cause noise in the boarding strategies, likely eroding some of the difference among their performances. Lastly, a strongly simplifying assumption is the issue of luggage stashing and seat obstruction, as sometimes passengers do not find space for their luggage and may hold up the queue if a certain area of seats has passengers all with large luggage, and alternatively, passengers may pass each other to their seats. without waiting for the other to stand up, etc. These simplifications impose a limitation on the validity of our results, however some of these were also used by Jafer & Mi (2017) and some may not matter for the outcome after all.

7 Conclusion

Based on our results in Table 3 after taking the average of all 100 simulations we find that the window middle aisle strategy is most effective boarding strategy due to the total boarding time, the CVAR 95% value and the mean individual boarding time being the lowest. The second best strategy is found to be the optimal boarding strategy which also

presents low values of total boarding time, the CVAR 95% value and the mean individual boarding time. The back to front strategy appears to be the least efficient boarding strategy according to the results of the simulation, as the total boarding time, the CVAR 95% value and the mean individual boarding time, present the highest values. The results are in line with the findings of Jafer & Mi (2017) who also found the window middle aisle strategy to be the fastest and the most efficient, despite some slight differences in the methodology of their study. However, this is not in line with Bachmat et al. (2009b), who found that the window-middle-aisle strategy was outperformed by the DOB strategy which allows passengers to board in an outside in fashion while respecting the back to front order.

8 Appendix

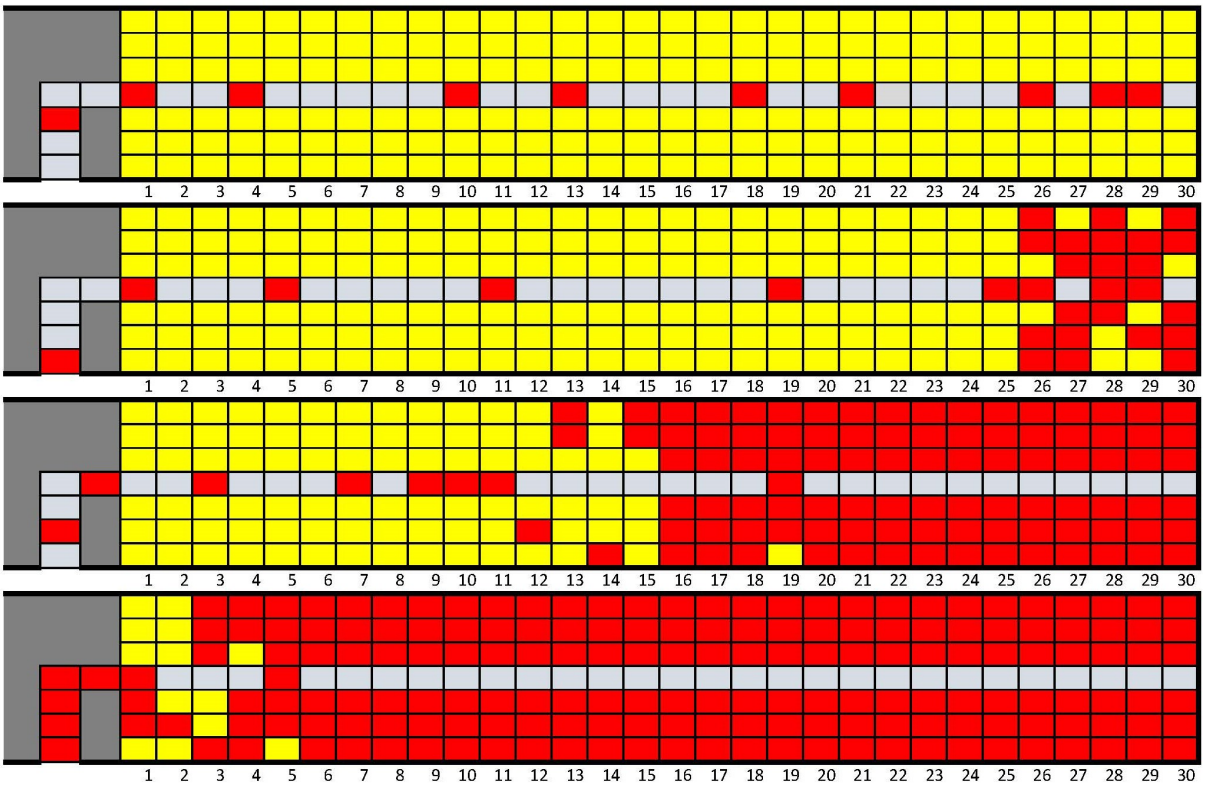

Figure 3: Back-to-front strategy snapshots of the boarding procedure (conceptual snapshots, not taken from our simulations). The dark grey area is inaccessible for passengers, light grey marks a free aisle cell, yellow marks an unoccupied seat and red marks cells occupied by a passenger.

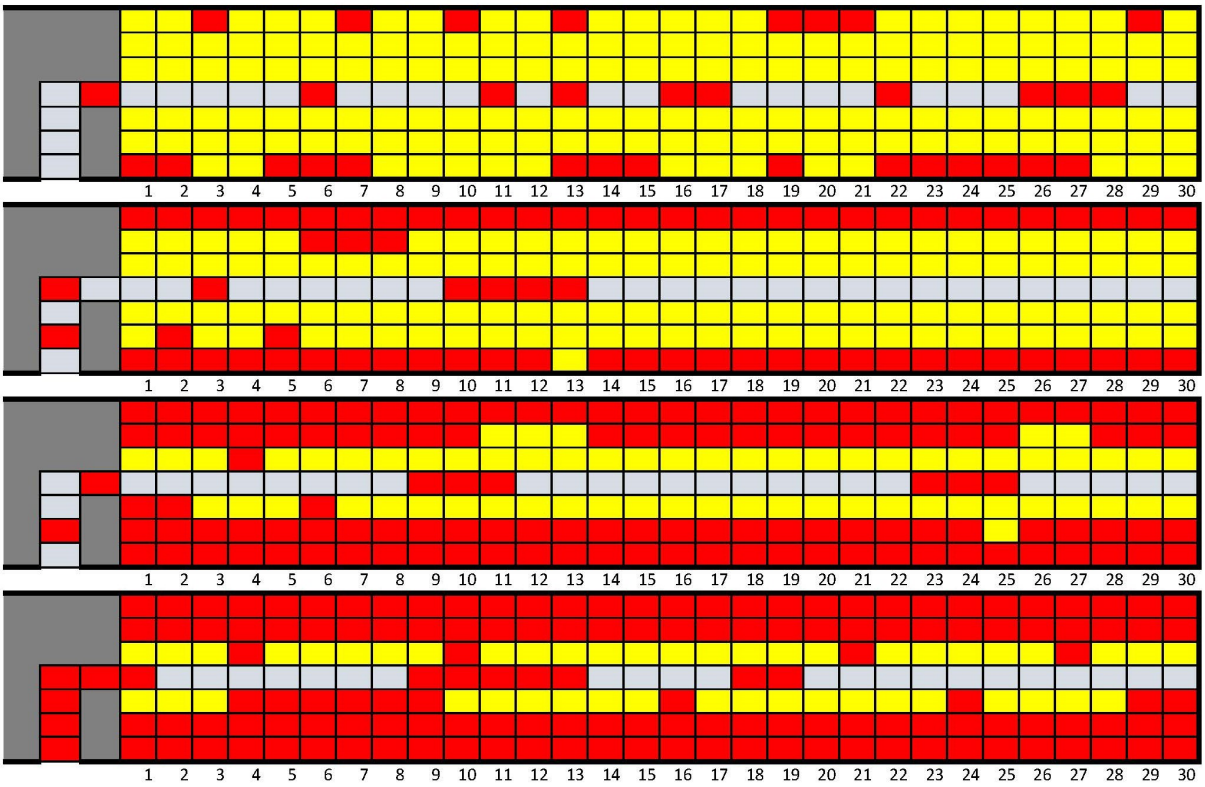

Figure 4: Window-Middle-Aisle strategy snapshots of the boarding procedure (conceptual snapshots, not taken from our simulations). The dark grey area is inaccessible for passengers, light grey marks a free aisle cell, yellow marks an unoccupied seat and red marks cells occupied by a passenger.

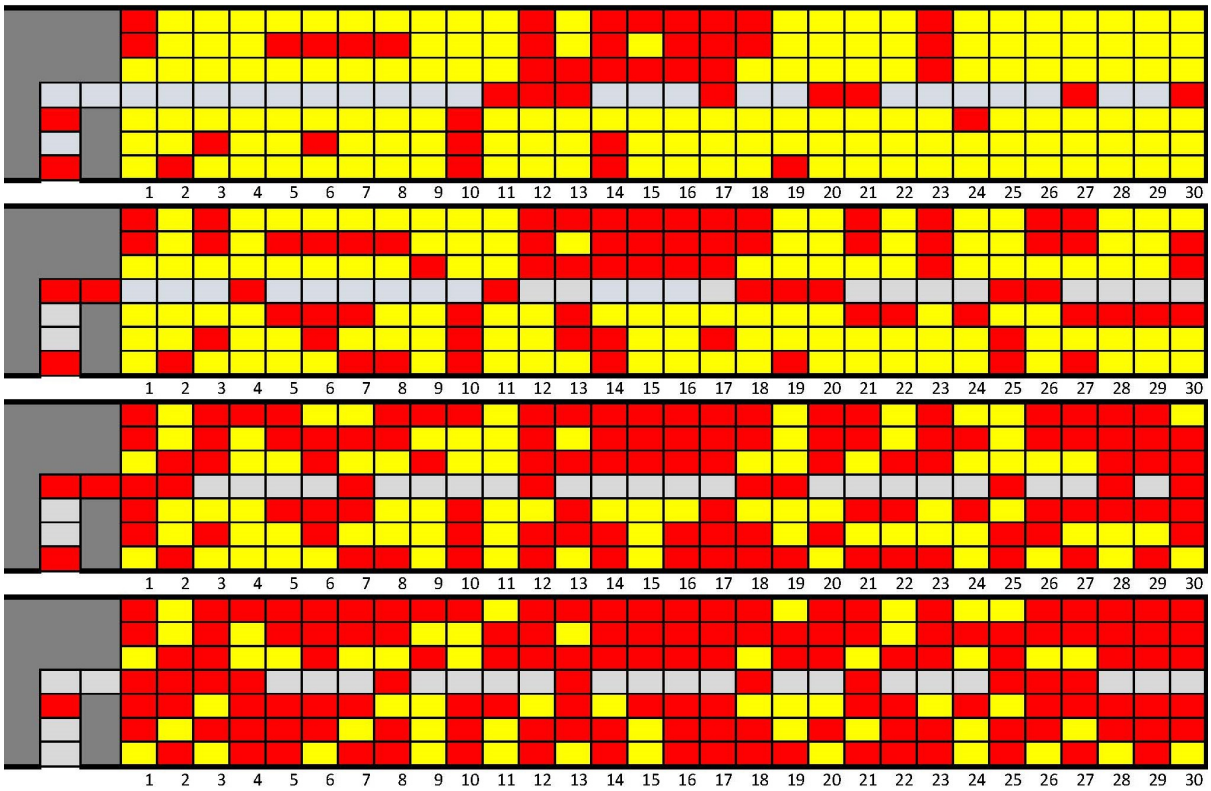

Figure 5: Random strategy snapshots of the boarding procedure (conceptual snapshots, not taken from our simulations). The dark grey area is inaccessible for passengers, light grey marks a free aisle cell, yellow marks an unoccupied seat and red marks cells occupied by a passenger.

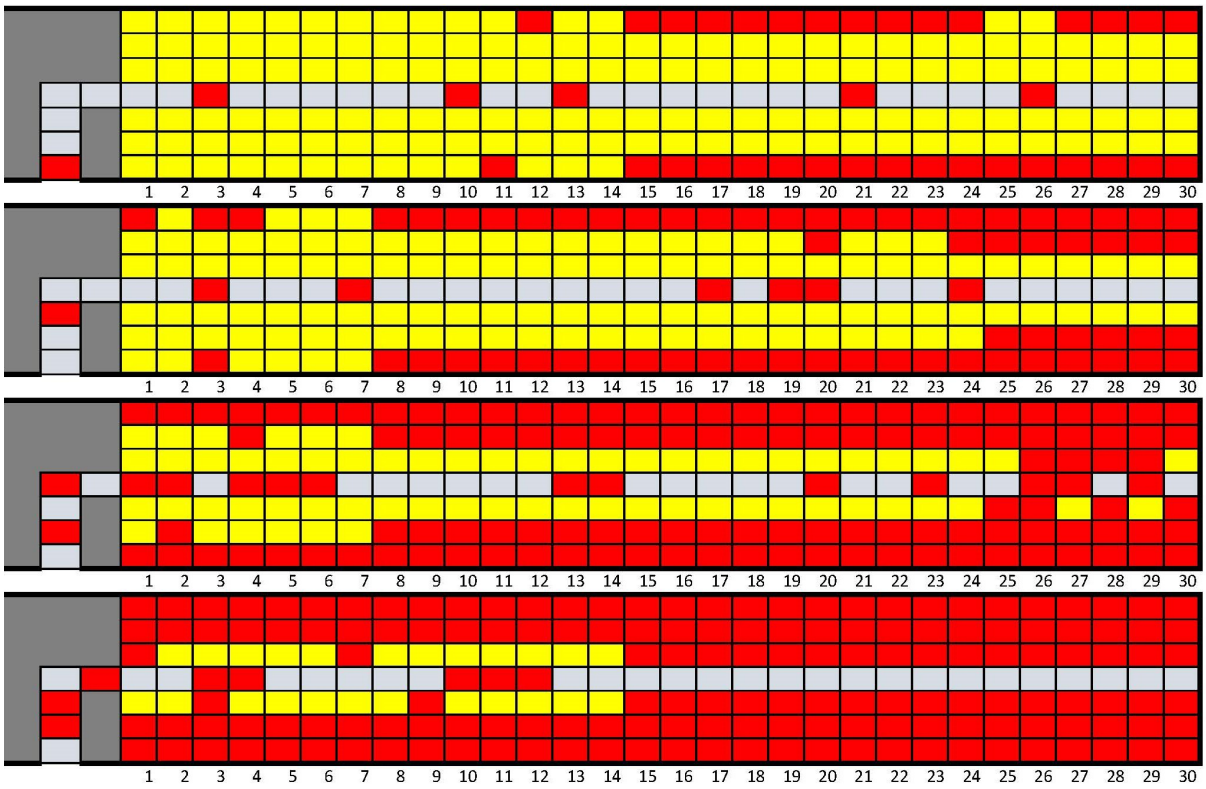

Figure 6: Reverse-Pyramid strategy snapshots of the boarding procedure (conceptual snapshots, not taken from our simulations). The dark grey area is inaccessible for passengers, light grey marks a free aisle cell, yellow marks an unoccupied seat and red marks cells occupied by a passenger.

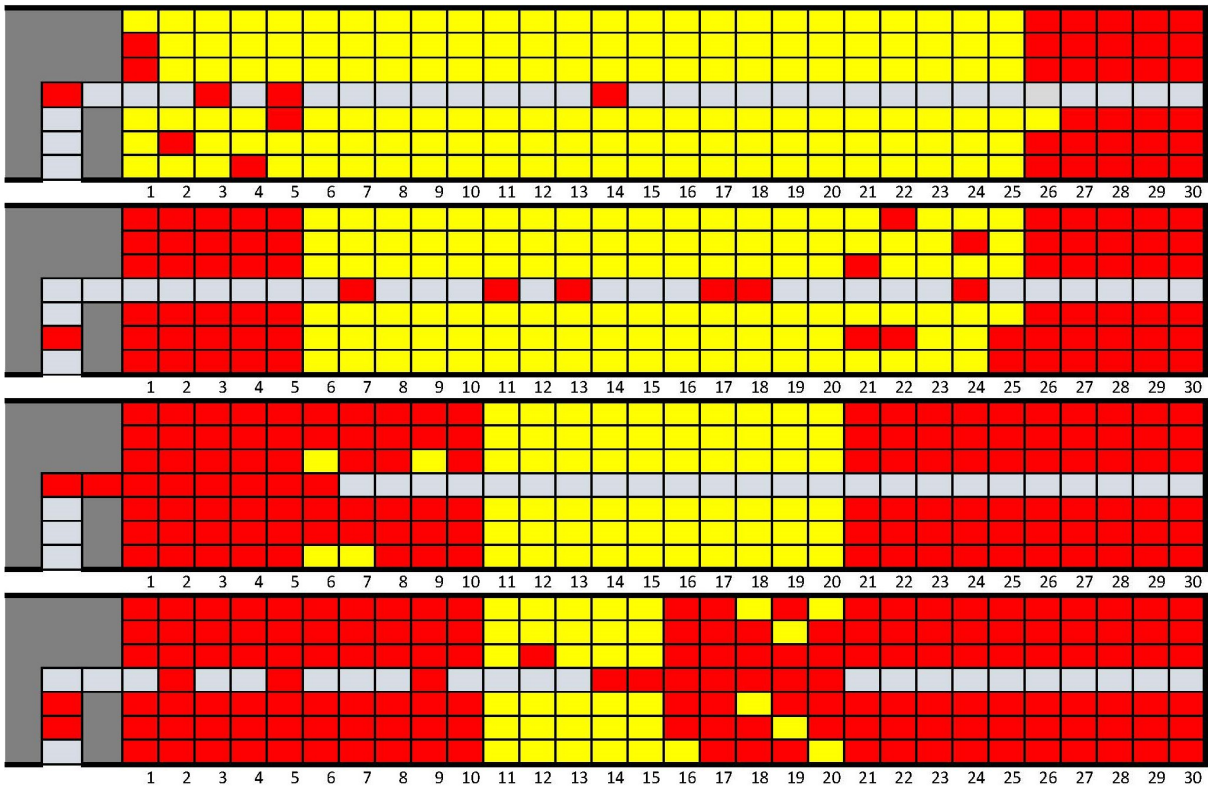

Figure 7: Rotating-zone strategy snapshots of the boarding procedure (conceptual snapshots, not taken from our simulations). The dark grey area is inaccessible for passengers, light grey marks a free aisle cell, yellow marks an unoccupied seat and red marks cells occupied by a passenger.

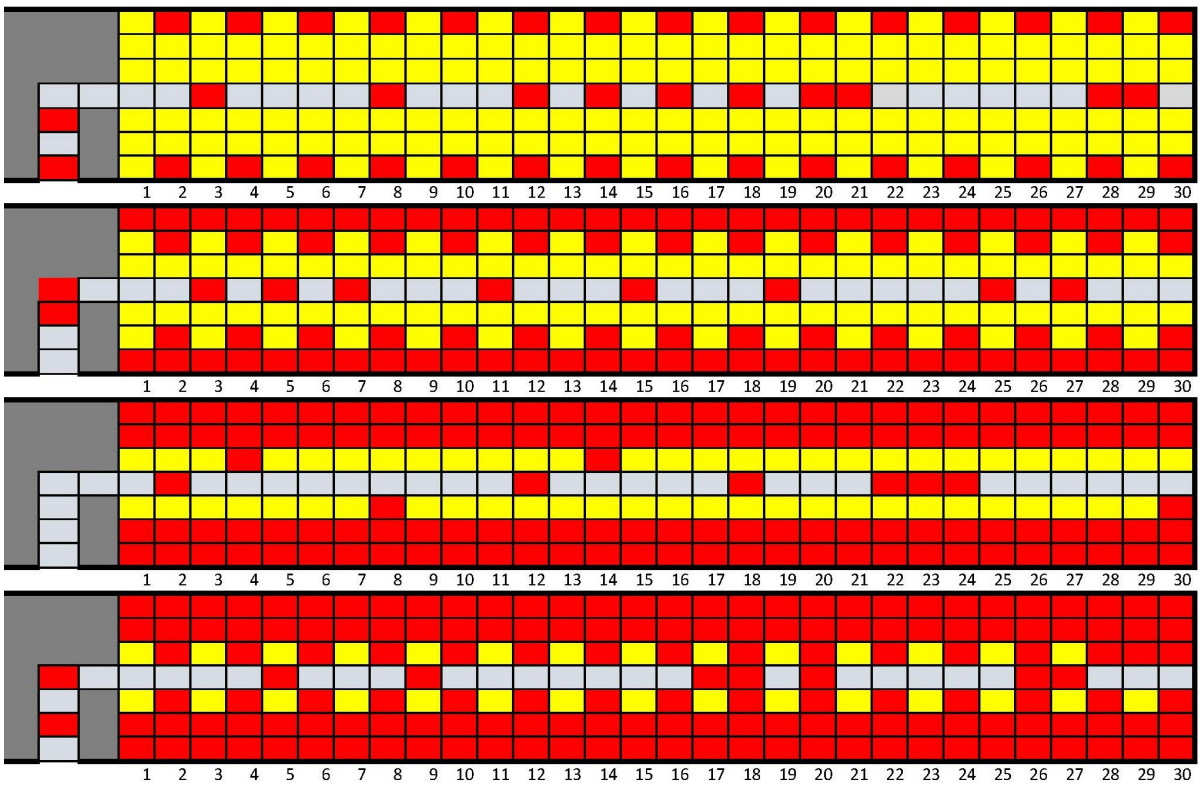

Figure 8: Optimal strategy snapshots of the boarding procedure (conceptual snapshots, not taken from our simulations). The dark grey area is inaccessible for passengers, light grey marks a free aisle cell, yellow marks an unoccupied seat and red marks cells occupied by a passenger.

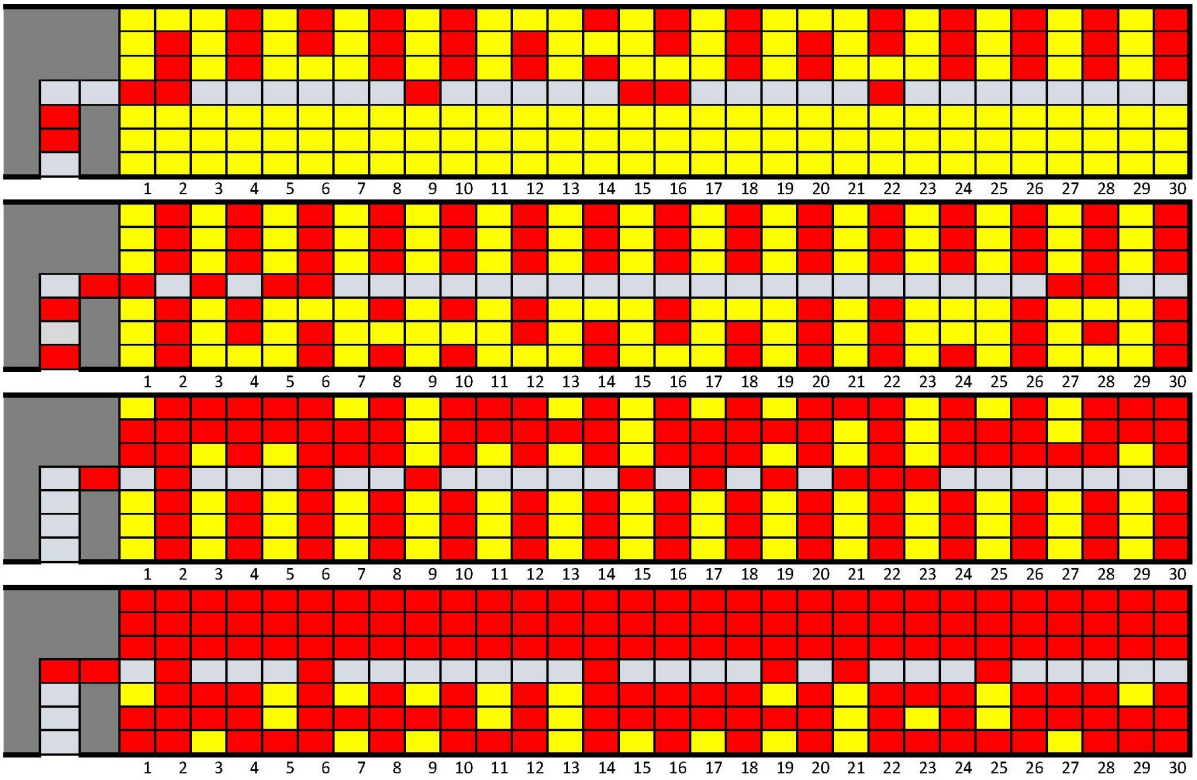

Figure 9: Practical-optimal strategy snapshots of the boarding procedure (conceptual snapshots, not taken from our simulations). The dark grey area is inaccessible for passengers, light grey marks a free aisle cell, yellow marks an unoccupied seat and red marks cells occupied by a passenger.

9 Code

Listing 1: Package imports, magic numbers and function calls

```
import numpy as np
import random
import time
from statistics import mean
import csv

random.seed(10)

######################################################################
### Magic numbers
# Number of simulations per boarding strategy
nsims = 100

# All analyzed boarding strategies
strategies = ["backtofront", "outsidein", "rotatingzone", "optimal", "pracoptimal", "revpyramid"]

# Plane attributes
# Number of seat rows on the plane
nrows = 30
# Seats per row
nseats = 6
# Total length of the aisle
l_aisle = 35

# The space occupied on the aisle is larger next to the seats than on the first part.
size_aisle_seats = 0.7112
size_aisle_begin = 0.4572

# Mean arrival delay (in seconds)
mu_arrival = 2

######################################################################
### Simulation      (initialization is done in the SimulationExecutor function)

# Run the simulation!
time_measures = Simulation_analysis(strategies, nrows, nseats, l_aisle, size_aisle_seats, size_aisle_begin, mu_arrival)
```

Listing 2: Boarding strategy function

```
def BoardingStrat(type):
    list_order = []
    if type == "backtofront":
        zone1_seats = []
        zone2_seats = []
        zone3_seats = []
        zone4_seats = []
        zone5_seats = []
        zone6_seats = []
        for row in [1, 2, 3, 4, 5]:
            for col in range(6):
                zone1_seats.append((row, col + 1))
        for row in [6, 7, 8, 9, 10]:
            for col in range(6):
                zone2_seats.append((row, col + 1))
        for row in [11, 12, 13, 14, 15]:
            for col in range(6):
                zone3_seats.append((row, col + 1))
        for row in [16, 17, 18, 19, 20]:
            for col in range(6):
                zone4_seats.append((row, col + 1))
        for row in [21, 22, 23, 24, 25]:
            for col in range(6):
                zone5_seats.append((row, col + 1))
        for row in [26, 27, 28, 29, 30]:
            for col in range(6):
                zone6_seats.append((row, col + 1))
        while bool(zone6_seats) is True:
            choice = random.sample(zone6_seats, 1)[0]
            remove = zone6_seats.index(choice)
            list_order.append(choice)
            zone6_seats.pop(remove)
        while bool(zone5_seats) is True:
            choice = random.sample(zone5_seats, 1)[0]
            remove = zone5_seats.index(choice)
            list_order.append(choice)
            zone5_seats.pop(remove)
        while bool(zone4_seats) is True:
            choice = random.sample(zone4_seats, 1)[0]
            remove = zone4_seats.index(choice)
            list_order.append(choice)
            zone4_seats.pop(remove)
        while bool(zone3_seats) is True:
            choice = random.sample(zone3_seats, 1)[0]
            remove = zone3_seats.index(choice)
            list_order.append(choice)
            zone3_seats.pop(remove)
```

```
        while bool(zone2_seats) is True:
            choice = random.sample(zone2_seats, 1)[0]
            remove = zone2_seats.index(choice)
            list_order.append(choice)
            zone2_seats.pop(remove)
        while bool(zone1_seats) is True:
            choice = random.sample(zone1_seats, 1)[0]
            remove = zone1_seats.index(choice)
            list_order.append(choice)
            zone1_seats.pop(remove)
    elif type == "outsidein":
        window_seats = []
        middle_seats = []
        aisle_seats = []
        for row in range(30):
            for col in [1,6]:
                window_seats.append((row + 1, col))
        for row in range(30):
            for col in [2,5]:
                middle_seats.append((row + 1, col))
        for row in range(30):
            for col in [3,4]:
                aisle_seats.append((row + 1, col))
        while bool(window_seats) is True:
            choice = random.sample(window_seats, 1)[0]
            remove = window_seats.index(choice)
            list_order.append(choice)
            window_seats.pop(remove)
        while bool(middle_seats) is True:
            choice = random.sample(middle_seats, 1)[0]
            remove = middle_seats.index(choice)
            list_order.append(choice)
            middle_seats.pop(remove)
        while bool(aisle_seats) is True:
            choice = random.sample(aisle_seats, 1)[0]
            remove = aisle_seats.index(choice)
            list_order.append(choice)
            aisle_seats.pop(remove)
    elif type == "rotatingzone":
        zone1_seats = []
        zone2_seats = []
        zone3_seats = []
        zone4_seats = []
        zone5_seats = []
        zone6_seats = []
        for row in [1,2,3,4,5]:
            for col in range(6):
```

```
                zone1_seats.append((row, col+1))
        for row in [6,7,8,9,10]:
            for col in range(6):
                zone2_seats.append((row, col+1))
        for row in [11,12,13,14,15]:
            for col in range(6):
                zone3_seats.append((row, col+1))
        for row in [16,17,18,19,20]:
            for col in range(6):
                zone4_seats.append((row, col+1))
        for row in [21,22,23,24,25]:
            for col in range(6):
                zone5_seats.append((row, col+1))
        for row in [26,27,28,29,30]:
            for col in range(6):
                zone6_seats.append((row, col+1))
        while bool(zone6_seats) is True:
            choice = random.sample(zone6_seats, 1)[0]
            remove = zone6_seats.index(choice)
            list_order.append(choice)
            zone6_seats.pop(remove)
        while bool(zone1_seats) is True:
            choice = random.sample(zone1_seats, 1)[0]
            remove = zone1_seats.index(choice)
            list_order.append(choice)
            zone1_seats.pop(remove)
        while bool(zone5_seats) is True:
            choice = random.sample(zone5_seats, 1)[0]
            remove = zone5_seats.index(choice)
            list_order.append(choice)
            zone5_seats.pop(remove)
        while bool(zone2_seats) is True:
            choice = random.sample(zone2_seats, 1)[0]
            remove = zone2_seats.index(choice)
            list_order.append(choice)
            zone2_seats.pop(remove)
        while bool(zone4_seats) is True:
            choice = random.sample(zone4_seats, 1)[0]
            remove = zone4_seats.index(choice)
            list_order.append(choice)
            zone4_seats.pop(remove)
        while bool(zone3_seats) is True:
            choice = random.sample(zone3_seats, 1)[0]
            remove = zone3_seats.index(choice)
            list_order.append(choice)
            zone3_seats.pop(remove)
    elif type == "optimal":
```

```
        zone1_seats = []
        zone2_seats = []
        zone3_seats = []
        zone4_seats = []
        zone5_seats = []
        zone6_seats = []
        zone7_seats = []
        zone8_seats = []
        zone9_seats = []
        zone10_seats = []
        zone11_seats = []
        zone12_seats = []
        for row in range(30, 0, -2):
            zone1_seats.append((row, 1))
        for row in range(30, 0, -2):
            zone2_seats.append((row, 6))
        for row in range(29, 0, -2):
            zone3_seats.append((row, 1))
        for row in range(29, 0, -2):
            zone4_seats.append((row, 6))
        for row in range(30, 0, -2):
            zone5_seats.append((row, 2))
        for row in range(30, 0, -2):
            zone6_seats.append((row, 5))
        for row in range(29, 0, -2):
            zone7_seats.append((row, 2))
        for row in range(29, 0, -2):
            zone8_seats.append((row, 5))
        for row in range(30, 0, -2):
            zone9_seats.append((row, 3))
        for row in range(30, 0, -2):
            zone10_seats.append((row, 4))
        for row in range(29, 0, -2):
            zone11_seats.append((row, 3))
        for row in range(29, 0, -2):
            zone12_seats.append((row, 4))
        while bool(zone1_seats) is True:
            choice = random.sample(zone1_seats, 1)[0]
            remove = zone1_seats.index(choice)
            list_order.append(choice)
            zone1_seats.pop(remove)
        while bool(zone2_seats) is True:
            choice = random.sample(zone2_seats, 1)[0]
            remove = zone2_seats.index(choice)
            list_order.append(choice)
            zone2_seats.pop(remove)
        while bool(zone3_seats) is True:
```

```
            choice = random.sample(zone3_seats, 1)[0]
            remove = zone3_seats.index(choice)
            list_order.append(choice)
            zone3_seats.pop(remove)
        while bool(zone4_seats) is True:
            choice = random.sample(zone4_seats, 1)[0]
            remove = zone4_seats.index(choice)
            list_order.append(choice)
            zone4_seats.pop(remove)
        while bool(zone5_seats) is True:
            choice = random.sample(zone5_seats, 1)[0]
            remove = zone5_seats.index(choice)
            list_order.append(choice)
            zone5_seats.pop(remove)
        while bool(zone6_seats) is True:
            choice = random.sample(zone6_seats, 1)[0]
            remove = zone6_seats.index(choice)
            list_order.append(choice)
            zone6_seats.pop(remove)
        while bool(zone7_seats) is True:
            choice = random.sample(zone7_seats, 1)[0]
            remove = zone7_seats.index(choice)
            list_order.append(choice)
            zone7_seats.pop(remove)
        while bool(zone8_seats) is True:
            choice = random.sample(zone8_seats, 1)[0]
            remove = zone8_seats.index(choice)
            list_order.append(choice)
            zone8_seats.pop(remove)
        while bool(zone9_seats) is True:
            choice = random.sample(zone9_seats, 1)[0]
            remove = zone9_seats.index(choice)
            list_order.append(choice)
            zone9_seats.pop(remove)
        while bool(zone10_seats) is True:
            choice = random.sample(zone10_seats, 1)[0]
            remove = zone10_seats.index(choice)
            list_order.append(choice)
            zone10_seats.pop(remove)
        while bool(zone11_seats) is True:
            choice = random.sample(zone11_seats, 1)[0]
            remove = zone11_seats.index(choice)
            list_order.append(choice)
            zone11_seats.pop(remove)
        while bool(zone12_seats) is True:
            choice = random.sample(zone12_seats, 1)[0]
            remove = zone12_seats.index(choice)
```

```
            list_order.append(choice)
            zone12_seats.pop(remove)
    elif type == "pracoptimal":
        zone1_seats = []
        zone2_seats = []
        zone3_seats = []
        zone4_seats = []
        for row in range(30, 0, -2):
            for col in [1, 2, 3]:
                zone1_seats.append((row, col))
        for row in range(30, 0, -2):
            for col in [4, 5, 6]:
                zone2_seats.append((row, col))
        for row in range(29, 0, -2):
            for col in [1, 2, 3]:
                zone3_seats.append((row, col))
        for row in range(29, 0, -2):
            for col in [4, 5, 6]:
                zone4_seats.append((row, col))
        while bool(zone1_seats) is True:
            choice = random.sample(zone1_seats, 1)[0]
            remove = zone1_seats.index(choice)
            list_order.append(choice)
            zone1_seats.pop(remove)
        while bool(zone2_seats) is True:
            choice = random.sample(zone2_seats, 1)[0]
            remove = zone2_seats.index(choice)
            list_order.append(choice)
            zone2_seats.pop(remove)
        while bool(zone3_seats) is True:
            choice = random.sample(zone3_seats, 1)[0]
            remove = zone3_seats.index(choice)
            list_order.append(choice)
            zone3_seats.pop(remove)
        while bool(zone4_seats) is True:
            choice = random.sample(zone4_seats, 1)[0]
            remove = zone4_seats.index(choice)
            list_order.append(choice)
            zone4_seats.pop(remove)
    elif type == "revpyramid":
        zone1_seats = []
        zone2_seats = []
        zone3_seats = []
        zone4_seats = []
        zone5_seats = []
        zone6_seats = []
        # Zone 1
```

```
        for row in range(15, 30):
            for col in [1, 6]:
                zone1_seats.append((row + 1, col))
        # Zone 2
        for row in range(7, 15):
            for col in [1, 6]:
                zone2_seats.append((row + 1, col))
        for row in range(23, 30):
            for col in [2, 5]:
                zone2_seats.append((row + 1, col))
        # Zone 3
        for row in range(0, 7):
            for col in [1, 6]:
                zone3_seats.append((row + 1, col))
        for row in range(15, 23):
            for col in [2, 5]:
                zone3_seats.append((row + 1, col))
        # Zone 4
        for row in range(23, 30):
            for col in [3, 4]:
                zone4_seats.append((row + 1, col))
        for row in range(7, 15):
            for col in [2, 5]:
                zone4_seats.append((row + 1, col))
        # Zone 5
        for row in range(0, 7):
            for col in [2, 5]:
                zone5_seats.append((row + 1, col))
        for row in range(15, 23):
            for col in [3, 4]:
                zone5_seats.append((row + 1, col))
        # Zone 6
        for row in range(0, 15):
            for col in [3, 4]:
                zone6_seats.append((row + 1, col))
        while bool(zone1_seats) is True:
            choice = random.sample(zone1_seats, 1)[0]
            remove = zone1_seats.index(choice)
            list_order.append(choice)
            zone1_seats.pop(remove)
        while bool(zone2_seats) is True:
            choice = random.sample(zone2_seats, 1)[0]
            remove = zone2_seats.index(choice)
            list_order.append(choice)
            zone2_seats.pop(remove)
        while bool(zone3_seats) is True:
            choice = random.sample(zone3_seats, 1)[0]
```

```
            remove = zone3_seats.index(choice)
            list_order.append(choice)
            zone3_seats.pop(remove)
        while bool(zone4_seats) is True:
            choice = random.sample(zone4_seats, 1)[0]
            remove = zone4_seats.index(choice)
            list_order.append(choice)
            zone4_seats.pop(remove)
        while bool(zone5_seats) is True:
            choice = random.sample(zone5_seats, 1)[0]
            remove = zone5_seats.index(choice)
            list_order.append(choice)
            zone5_seats.pop(remove)
        while bool(zone6_seats) is True:
            choice = random.sample(zone6_seats, 1)[0]
            remove = zone6_seats.index(choice)
            list_order.append(choice)
            zone6_seats.pop(remove)
    else:  # random
        seats = []
        for row in range(30):
            for col in range(6):
                seats.append((row+1, col+1))
        while bool(seats) is True:
            choice = random.sample(seats, 1)[0]
            remove = seats.index(choice)
            list_order.append(choice)
            seats.pop(remove)
    return list_order
```

Listing 3: Functions developed for the simulation

```
def Queue_former(strategy):
    arrival_queue = BoardingStrat(strategy)
    passengers = []
    ID = 0
    for i in enumerate(arrival_queue):
        ID += 1
        passengers.append(PassengerGenerator(ID, i[1][0], i[1][1]))
    return passengers



def SeatingManager(arriving_passenger, seating_matrix, current_time):
    seat = arriving_passenger["seat"]
    row = arriving_passenger["row"]
```

```
    if seat==3 or seat==4:
        # Arriving has aisle seat
        aisle_seating = random.uniform(2, 5)  # Aisle seating time (aisle can be taken regardless of people already sitting in the row
        arriving_passenger["timespent"]["seating_time"] += aisle_seating
        seating_time = aisle_seating
    elif seat==2 or seat==5:
        # Arriving has middle seat
        if seat==2:
            target_seat = 3
        else:
            target_seat = 4
        if bool(seating_matrix[row-1][target_seat-1]) is True:
            # Aisle seat is taken
            aisle_seating = random.uniform(2, 5)  # Standup time of the person in the aisle seat
            middle_seating = random.uniform(3, 6)  # middle seating time
            seating_matrix[row-1][target_seat-1]["timespent"]["standup"] += aisle_seating*2 + middle_seating
            arriving_passenger["timespent"]["seating_time"] += middle_seating+aisle_seating
            seating_time = aisle_seating*2 + middle_seating
        else:
            middle_seating = random.uniform(3, 6)  # middle seating time
            arriving_passenger["timespent"]["seating_time"] += middle_seating
            seating_time = middle_seating
    else:
        # Arriving has window seat
        if seat==1:
            target_middle = 2
            target_aisle = 3
        else:
            target_middle = 5
            target_aisle = 4
        if bool(seating_matrix[row - 1][target_aisle - 1]) is True:
            if bool(seating_matrix[row - 1][target_middle - 1]) is True:
                # Both aisle and middle are taken
                aisle_seating = random.uniform(2, 5)  # Standup time of the person in the aisle seat
                middle_seating = random.uniform(3, 6)  # Standup time of the person in the middle seat
                window_seating = random.uniform(4, 7)  # window seating time
                arriving_passenger["timespent"]["seating_time"] += window_seating + aisle_seating + middle_seating
```

```
                seating_matrix[row - 1][target_aisle -
                    1]["timespent"]["standup"] += aisle_seating*2 +
                    middle_seating*2 + window_seating
                seating_matrix[row - 1][target_middle -
                    1]["timespent"]["standup"] += middle_seating * 2 +
                    window_seating
                seating_time = aisle_seating*2 + middle_seating*2 +
                    window_seating
            else:
                # Only aisle seat is taken
                aisle_seating = random.uniform(2, 5)  # Standup time of
                    the person in the aisle seat
                window_seating = random.uniform(4, 7)  # window seating
                    time
                seating_matrix[row - 1][target_aisle -
                    1]["timespent"]["standup"] += aisle_seating * 2 +
                    window_seating
                arriving_passenger["timespent"]["seating_time"] +=
                    window_seating + aisle_seating
                seating_time = aisle_seating * 2 + window_seating
        else:
            if bool(seating_matrix[row - 1][target_middle - 1]) is True:
                # Only middle seat is taken
                middle_seating = random.uniform(3, 6)  # Standup time of
                    the person in the middle seat
                window_seating = random.uniform(4, 7)  # window seating
                    time
                seating_matrix[row - 1][target_middle -
                    1]["timespent"]["standup"] += middle_seating * 2 +
                    window_seating
                arriving_passenger["timespent"]["seating_time"] +=
                    window_seating + middle_seating
                seating_time = middle_seating * 2 + window_seating
            else:
                window_seating = random.uniform(4, 7)  # window seating
                    time
                arriving_passenger["timespent"]["seating_time"] +=
                    window_seating
                seating_time = window_seating

    # Setting time measures.
    aisle_occupancy_time =
    ↪   arriving_passenger["timespent"]["luggage_stash"] + seating_time
    arriving_passenger["timespent"]["time_seated"] = current_time[0] +
    ↪   arriving_passenger["timespent"]["luggage_stash"] +
    ↪   arriving_passenger["timespent"]["seating_time"]
    arriving_passenger["timespent"]["board_time_cum"] =
    ↪   arriving_passenger["timespent"]["time_seated"] -
    ↪   arriving_passenger["timespent"]["time_entrance"]
```

```
79
80      # Seat the arriving passenger in the seating matrix
81      seating_matrix[row - 1][seat - 1] = arriving_passenger
82
83      # Return the seating matrix, the time it takes for the aisle to
        ↪  clear.
84      res_seating = [seating_matrix, aisle_occupancy_time]
85      return res_seating
86
87
88
89  def PassengerGenerator(ID, row, seat):
90      # Simulated parameters
91      has_luggage = rnd.choice((1, 0), p=[0.8, 0.2])  # logical if person
        ↪  has a luggage (1-yes, 0-no)
92      if has_luggage==1:
93          luggage_stash = random.uniform(6, 30)  # seconds spent with
            ↪  luggage stash
94      else:
95          luggage_stash = 0
96      walkspeed = random.uniform(0.27, 0.44)  # meter per second
97
98      # Calculated attributes and containing these attributes
99      walking_time = (0.7112/walkspeed) * row + 4*(0.4572/walkspeed)
100     dict_res = {
101         "row": row,
102         "seat": seat,
103         "has_luggage": has_luggage,
104         "walkspeed": {'aisle_begin' : size_aisle_begin/walkspeed,
105                       'aisle_seats' : size_aisle_seats/walkspeed},
106         "aisle_pos": None,
107         "seated": False,
108         "ID": ID,
109         "timespent": {
110             "board_time_cum": None,
111             "time_entrance": None,
112             "time_seated": None,
113             "walking": walking_time,
114             "waiting": 0,
115             "luggage_stash": luggage_stash,
116             "seating_time": 0,
117             "standup": 0
118         }
119     }
120     return dict_res
121
122
```

```
123
124 def New_arrival(passengers, aisle, current_time, events):
125     # We find the first passenger on the passengers list that has not
        ↪ yet been assigned a spot in the aisle.
126     for i in range(len(passengers)):
127         if passengers[i]['aisle_pos']==None:
128             while aisle[0] == 0:
129                 passengers[i]['aisle_pos'] = 0
130                 passengers[i]['timespent']['time_entrance'] =
                    ↪ current_time[0]
131                 aisle[0] = 1
132                 # Make a new entry for the passenger in the event list
133                 pass_event = Passenger_event(passengers[i]['ID'],
                    ↪ passengers[i]['walkspeed']['aisle_begin'])
134                 events.append(pass_event)
135
136
137
138 def Passenger_event(ID, timer):
139     return{'ID': ID, 'timer' : timer,}
140
141
142
143 def BoardingSimulator9000(current_time, passengers, aisle, events,
    ↪ seating_matrix, mu_arrival):
144     # Step 1: Identify the next event
145     min_timer = min(events, key = lambda x: x['timer'])['timer']    #
        ↪ Find the time to the next event
146     # Retrieve the event list index of the next event (by matching the
        ↪ smallest timer)
147     action_index = next((index for (index, e) in enumerate(events) if
        ↪ e["timer"] == min_timer), None)
148
149     current_time[0] += min_timer # Update current time
150     for i in range(len(events)):
151         events[i]['timer'] -= min_timer     # Update the event timers.
152
153     # Step 2: Execute the next event
154     # First we define what happens if the event is a new arrival
155     if events[action_index]['ID'] == 'Arrival':
156         # In case the first spot on the aisle is occupied, the arrival
            ↪ is delayed.
157         if bool(aisle[0]):
158             events[action_index]['timer'] = 1   # Setting the delay
                ↪ arbitrarily to 1 second
159             # Note that we are only counting waiting time on the plane.
                ↪ Waiting time in the tube is
```

```
        else:
            events[action_index]['timer'] =
             ↪ random.expovariate(1/mu_arrival) # Otherwise draw from
             ↪ an exp. distribution with mean 2
            New_arrival(passengers, aisle, current_time, events)    #
             ↪ And let the new passenger arrive.
        # If all passengers are in the plane, remove the arrival event
         ↪ from the event list.
        if all(p['aisle_pos'] is not None for p in passengers):
            del events[action_index]

    else: # If we are not dealing with a new arrival, we must be dealing
     ↪ with a passenger.
        # Designate the passenger that is going to move.
        pass_index = next((index for (index, d) in enumerate(passengers)
         ↪ if d["ID"] == events[action_index]['ID']), None)
        # Retrieve the passenger's position on the aisle.
        aisle_position = passengers[pass_index]['aisle_pos']

        # First we check if the passenger has already been seated. If
         ↪ so, we remove the passenger's actions from the event list.
        if passengers[pass_index]['seated'] == True:
            del events[action_index]
            # Uncomment line below for continuously keeping track of the
             ↪ simulation.
            # print(f" Passenger {passengers[pass_index]['ID']} is
             ↪ seated at time {current_time[0]}.")
            # And his space on the aisle is cleared.
            aisle[aisle_position] = 0

        # If not yet seated, the passenger will try to move up a space
         ↪ on the aisle.
        else:
            # If the next space on the aisle is occupied, the passenger
             ↪ needs to wait.
            if bool(aisle[aisle_position + 1]):
                # Taking into account smaller tiles at the beginning of
                 ↪ the aisle.
                if aisle_position < l_aisle - nrows:
                    # Schedule the passenger's next movement.
                    events[action_index]['timer'] =
                     ↪ passengers[pass_index]['walkspeed']['aisle_begin']
                    # Update waiting time.
                    passengers[pass_index]['timespent']['waiting'] +=
                     ↪ passengers[pass_index]['walkspeed']['aisle_begin']
                # Now for the larger aisle tiles at the seats.
                else:
```

```
193                     events[action_index]['timer'] =
                         ↪  passengers[pass_index]['walkspeed']['aisle_seats']
194                     passengers[pass_index]['timespent']['waiting'] +=
                         ↪  passengers[pass_index]['walkspeed']['aisle_seats']
195
196             # If the next space on the aisle is vacant, the passenger
                 ↪  moves.
197             else:
198                 aisle[aisle_position] = 0
199                 aisle[aisle_position + 1] = 1
200                 passengers[pass_index]['aisle_pos'] = aisle_position + 1
201
202                 # If he has not arrived at his row number yet, we simply
                     ↪  schedule his next movement.
203                 if not passengers[pass_index]['aisle_pos'] ==
                     ↪  passengers[pass_index]['row'] + l_aisle - nrows - 1:
204                     # Again accounting for different tile sizes. Some
                         ↪  duplicate code cannot be avoided here.
205                     if aisle_position < l_aisle - nrows:
206                         events[action_index]['timer'] =
                             ↪  passengers[pass_index]['walkspeed']['aisle_begin']
207                     else:
208                         events[action_index]['timer'] =
                             ↪  passengers[pass_index]['walkspeed']['aisle_seats']
209
210                 # Now for something very important: if the passenger
                     ↪  arrives at the row of his seat he needs to sit!
211                 else:
212                     # Next time this passenger's event occurs, the
                         ↪  passenger is seated and removed from the event
                         ↪  list.
213                     # Here we already mark this passenger as seated.
214                     passengers[pass_index]['seated'] = True
215                     res_seating = SeatingManager(passengers[pass_index],
                         ↪  seating_matrix, current_time)
216                     seating_matrix = res_seating[0]
217                     events[action_index]['timer'] = res_seating[1]
218
219
220
221 def SimulationExecutor(strategy, nrows, nseats, l_aisle,
    ↪  size_aisle_seats, size_aisle_begin, mu_arrival):
222     # Initialization
223     current_time = [0]                              # Initializing
         ↪  global time. For some reason this cannot simply be an integer,
         ↪  otherwise it does not update (took me forever to find this).
224     aisle = np.zeros(l_aisle)                       # The aisle is a
         ↪  binary vector, where '1' designates occupancy and '0' vacancy.
```

```
    seating_matrix = np.full((nrows,nseats), {})    # Initializing seating matrix
    passengers = Queue_former(strategy)             # Initializing the passengers according to the boarding strategy.
    events = [{'ID': 'Arrival', 'timer' : random.expovariate(1/mu_arrival)}]    # Initializing the event list.
    # At first the only active event is that of the next arrival.
    # As passengers enter the plane, their next action gets its separate event.

    # Execution
    while len(events)>0:                            # Once everybody is seated there are no more scheduled events and the simulation is done.
        BoardingSimulator9000(current_time, passengers, aisle, events, seating_matrix, mu_arrival)

    return seating_matrix



def Simulation_analysis(strategies, nrows, nseats, l_aisle, size_aisle_seats, size_aisle_begin, mu_arrival):

    # For each strategy and for each time measure I want a list that states the measured times for all passengers if all iterations.
    # So for 100 iterations each list should contain 180*100 = 18000 observations.

    # Setting up output structure
    # Running simulation once for setting up the output structure.
    seating_matrix = SimulationExecutor(strategies[0], nrows, nseats, l_aisle, size_aisle_seats, size_aisle_begin, mu_arrival)
    passenger_list = seating_matrix.reshape(-1)
    time_measures = []
    for strat in enumerate(strategies):
        keys = [key for key in passenger_list[0]['timespent']]
        stats = ['95thpercentile', 'mean']
        keys0 = stats + ['Total_boarding_time']
        keys.extend(keys0)
        time_measures.append({strat[1] : {key : [] for key in keys}})
        for stat in stats:
                time_measures[strat[0]][strat[1]][stat] = {key: [] for key in keys if key not in keys0}


```

```
259         # Saving output to output structure.
260         start = time.time()
261         for i in range(nsims):
262             seating_matrix = SimulationExecutor(strat[1], nrows, nseats,
                ↪  l_aisle, size_aisle_seats, size_aisle_begin, mu_arrival)
263             passenger_list = seating_matrix.reshape(-1)
264             for key in keys:
265                 if key not in keys0:
266                     key_list = [p['timespent'][key] for p in
                        ↪  passenger_list]
267 
                        ↪  time_measures[strat[0]][strat[1]][key].extend(key_list)
268 
269                     key_mean = mean(key_list)
270                     key_95percentile = np.percentile(key_list, 95)
271 
272 
                        ↪  time_measures[strat[0]][strat[1]]['mean'][key].extend([key_mean])
273 
                        ↪  time_measures[strat[0]][strat[1]]['95thpercentile'][key].extend([key_95percentile])
274 
                ↪  time_measures[strat[0]][strat[1]]['Total_boarding_time'].append(max(p['timespent']['time_seated'] for p in passenger_list))
                ↪  key = lambda x:
                ↪  x['timespent']['time_seated'])['timespent']['time_seated'])
275 
276         finish = time.time() - start
277         print(f" Running {nsims} simulations of boarding strategy
            ↪  {strat[1]} was completed in {finish} seconds.")
278 
279     return time_measures
```

References

Bachmat, E., Berend, D., & Sapir, L. (2009a). Analysis of airplane boarding times. *Aerospace*, *2*, 34. doi: 10.1287/opre.1080.0630

Bachmat, E., Berend, D., & Sapir, S., L.and Skiena. (2009b). A dynamically optimized aircraft boarding strategy. *Aerospace*, *58*, 7. doi: 10.1016/j.jairtraman.2016.10.010

Chun, H. W., & Wai Tak Mak, R. (1999). Reducing passenger boarding time in airplanes:a simulation based approach. *Aerospace*, *23*, 11. doi: 1094âĂŞ6977/99

Ferrari, P., & Nagel, K. (2005). Robustness of efficient passenger boarding strategies for airplanes. *Aerospace*, *1915*, 10. doi: 10.3141/1915-06

Jafer, S., & Mi, W. (2017, 11). Comparative study of aircraft boarding strategies using cellular discrete event simulation. *Aerospace*, *4*, 57. doi: 10.3390/aerospace4040057

Qiang, S., Jia, B., Qingxia, H., & Jiang.R. (2018). Simulation of free boarding process using a cellular automaton model for passenger dynamics. *Aerospace*, *15*, 11. doi: 10.1007/s11071-017-3867-5

Schultz, M. (2008). Implementation and application of a stochastic aircraft boarding model. *Aerospace*, *15*, 11. doi: 10.1016/2018.03.016

Van Landeghem, H., & Beuselinck, A. (2002). Reducing passenger boarding time in airplanes:a simulation based approach. *Aerospace*, *1915*, 14. doi: 10.1016/S0377-2217(01)00294-6