Contents

Letter to airline executive

Dear Sir/Mdm,

Thank you for giving us the opportunity to contribute in improving your airline's operating procedures. Over the past days, our team has decided to tackle this challenge; we have developed a series of robust models that simulate and model the boarding and disembarking procedures from a holistic perspective.

Throughout our development process, we have established a concrete set of desirables that provide insight into what passengers find convenient and what govern effective procedures. We thus develop a model that simulates individual passenger behavior and understands how they will act during boarding and disembarking their flight. The commonplace boarding procedures in question include *Random*, *By Seat*, and *By Section*. Ultimately, we aim to highlight techniques to streamline the boarding and disembarking processes, focusing specifically on ones which achieve a balance between time-efficiency and feasibility from a passenger perspective.

In the process of constructing a boarding time prediction model, we have identified several important factors. The first three factors are related to passenger demographics. First is walking speed, which is dependent largely on the age of the passenger. Second is sitting and standing speed, which influences the amount of accumulated time spent navigating around physical and human obstructions (e.g. when a passenger attempts to enter a row that is partially occupied). Third is the amount of luggage, which varies with the time a passenger spends obstructing the path of others in the aisle.

To make the model more realistic, we also considered customer satisfaction with the boarding procedure in use as a factor. We refer to the inclination of passengers to resist direction as reneging, which may occur at any phase of boarding and disembarking. Given the pandemic situation, we included capacity and luggage limitations in the model to improve its adaptability.

With these factors in mind, we simulated a stochastic process that mirrored the action of passengers boarding an aircraft either randomly or by prescribed methods. Simulations were run on various passenger aircraft models, such as Narrow Body; Flying Wing; and Two-Entrance, Two-Aisle. This allowed us to estimate the time-efficiency of various boarding and disembarking procedures. In particular, for the typical "Narrow Body" passenger aircraft model, we found that the *By Seat* method of boarding was fastest with an average elapsed time of 32.6 minutes. Furthermore, the performance of other boarding procedures are described in detail in the enclosed paper.

Thank you again for your interest. We look forward to your feedback.

Best,

IM^{2}C Team US-11617

I Introduction

1.1 Background

In commercial passenger air travel, airlines use various boarding and disembarking methods from completely unstructured (passengers board or leave the plane without guidance) to structured (passengers board or leave the plane using a prescribed method). Prescribed methods may be based on row numbers, seat positions, or priority groups. In practice, however, even when the prescribed method is announced, not all passengers follow the instructions.

The boarding process includes the movement of passengers from the entrance of the aircraft to their assigned seats. This movement can be hindered by aisle and seat interference. For example, many passengers have carry-on bags which they stow into the overhead bins before taking their seats. Each time a passenger stops to stow a bag, the queue of other passengers stops because narrow aircraft aisles allow only one passenger to pass at a time. Another hindrance is that some seats (e.g., window seats) are unreachable if other seats (e.g., aisle seats) are already occupied. When this occurs, some passengers must stand up and move into the aisle so other passengers can reach their seats.

The disembarking process is the opposite of boarding with its own possible hindrances to passenger movement. Some passengers are simply slow getting out of their seat and row, or slow moving to the exit. Passengers also block the aisle while collecting their belongings from either their seat or from the overhead bin forcing passengers behind them in the aircraft to wait.

1.2 Problem Restatement

In this paper, we aim to address the problem of modeling, optimizing, and analyzing boarding and disembarking procedures. In particular, we must consider consider how fluctuations in passenger attendance, luggage, and aircraft design influence these processes. Following such analyses, we must propose new procedures to decrease total time incurred by current methods.

II Assumptions & Variables

2.1 Assumptions

1. **After arriving at their seat during boarding, passengers do not stand back up unless they need to allow another passenger to reach their seat.**
During the boarding process, as passengers gradually arrive, passengers that have already sat down will not look to actively leave their seat. Unless they need to step into the aisle to let another passenger reach their seat, the hindrance of navigating throughout the packed aircraft will disincentivize passengers from repeatedly standing up.

2. **Passengers sit in the seat indicated on their ticket.**
Due to the high costs of airline tickets and the potential risk of getting removed off the aircraft, passengers do not actively attempt to steal another passenger's seat. Accordingly, they only look for their designated seat when boarding.

3. **The total number of carry-on items does not exceed the airliner's capacity.**
While some passengers may hold more carry-on items and completely fill their overhead bins, there is enough room elsewhere on the aircraft to accommodate all luggage.

4. **Aisles and seats are only wide enough for one person, and passengers do not trample past each other.** Given the compact nature of most airliners, it is reasonable to assume passengers maintain enough personal space, especially given current pandemic conditions. Unpredictable chaos may begin if passengers begin to pass and trample past each other in aisles and rows.

5. **Passengers boarding or disembarking the aircraft do not walk against the flow of traffic.** For our models, we assume passengers only walk in one direction towards the seat designated on their ticket or the exit (depending on boarding or disembarking procedure).

2.2 Variables

Table 1: Variables Used and Definitions

Symbol	Description
S	Set of all seats on an aircraft
$\mathcal{P}$	A partition of the set S (A boarding procedure).
w	An individual's walking speed
p_r	Reneging Rate

III Boarding Simulation Model

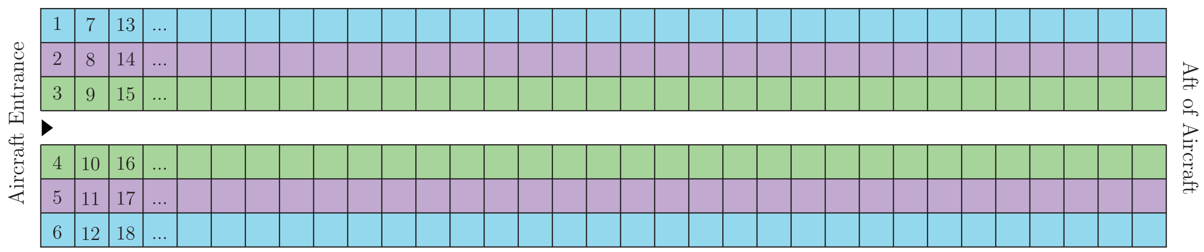
In order to evaluate and optimize the airline boarding process, we first develop an adaptable stochastic simulation of boarding. We accomplish this by concretely formulating what it means to be a generalized boarding (or disembarking) procedure. This approach allows us to test and predict the boarding of various existing and newly proposed boarding procedures. We consider the following factors that influence boarding time at the individual passenger level: walking speed, aircraft layout, passenger demographics & characteristics, luggage, and reneging rate. We do not consider sweeping factors such as weather or airline location, as these fluctuate heavily and do not immediately influence the success of one specific boarding procedure on general aircraft boarding.

3.1 What are Practical Boarding Procedures?

At its heart, any generalized boarding procedure partitions the set of passengers into distinct *boarding groups*. For any aircraft, suppose we number each seat s_i with a distinct integral value, indicating a seat number i. For l total seats, we have $|S| = l$, where $S = \{s_1, s_2, ..., s_l\}$ is the set of all seats on an aircraft. Suppose we partition S into n disjoint subsets $\mathcal{P} = \{S_1, S_2, ..., S_n\}$. We have:

$$S = \bigcup_{i=1}^{n} S_i \quad \text{where} \quad S_k \cap S_j = \varnothing \quad \text{for all } (k, j)$$

The key observation in making a general boarding procedure involves accounting for **all** seats on an aircraft. This is important due to the inherent fluctuations in passenger turn-up and the need for standardized training for airport staff— while different aircrafts may have different boarding procedures, it is unreasonable to have differing procedures for every individual flight.



$$S = \{1, 2, 3, 4, 5, 6, 7, 8, 9, ...\}$$

$$\mathcal{P} = \left\{ \underbrace{\{1, 6, 7, 12, 13, 18\}}_{\text{Boarding Group \#1}}, \underbrace{\{2, 5, 8, 11, 14, 17, ...\}}_{\text{Boarding Group \#2}}, \underbrace{\{3, 4, 9, 10, 15, 16, ...\}}_{\text{Boarding Group \#3}} \right\}$$

Figure 1: Partitions for a *Boarding by Seat* procedure in a “Narrow Body” Aircraft

We recognize that while many boarding procedures provide concrete theoretic improvements to boarding time, there is a fundamental factor that inhibits these propositions from hitting airports worldwide: the number of boarding groups n. With more boarding groups, passengers are naturally more inclined to resist staff instruction, regardless of the potential time-saves.

Having only one boarding group would be ideal for passengers, as it means there is no buffer time in between loading. While this doesn't always correspond with *quick* boarding, it is practically ideal for passengers. Conversely, having a boarding group for every single individual seat would be of least appeal to passengers. For an aircraft with l seats, the least practical solution would consist of l boarding groups. In the next section, we consider how this intuition relates to passenger behavior during boarding.

3.2 Passenger Reneging

Following a successful partition of seats, a boarding procedure involves airline staff calling up each boarding group individually. After one group fully boards, the next group will follow suit and begin boarding. Note that while a boarding group will contain all seat numbers in that group, it makes no prescription as to the order of passengers. That is to say, boarding groups enter completely randomized; passengers may act unpredictably and ignore the prescribed group assignments— a person in Boarding Group 2 may enter during the boarding of Group 3. We consider this behavior as *reneging*.

Passengers are naturally more likely to renege if the number of boarding groups is high— if the boarding procedure seems impractical. It is not uncommon to be rushed, impatient, and thus disorderly in such a situation. To account for this relationship, we consider a ***Reneging Rate*** (probability of reneging) that differs based on the number of boarding rounds in a procedure. According to the above description, a logistic function [8], or S-curve, can be used to formulate this relationship:

$$p_r = \frac{p_{r_1} p_{r_2} \exp(R(n-2))}{p_{r_2} + p_{r_1}(\exp(R(n-2)) - 1)} \tag{3.2.1}$$

Where n indicates the number of rounds in a procedure, p_{r_1} is the initial reneging rate with $n = 2$ groups[1], p_{r_2} is the maximal reneging rate, and R is a relative growth rate of the curve. As we will detail in our analysis, we set these parameters to $p_{r_1} = 0.1$, $p_{r_2} = 0.8$, $R = 0.4$ and provide a corresponding sensitivity analysis. Figure 3 reflects the above relationship.

[1]Observe that by definition $p_r = 0$ for $n = 1$ group, but all other group sizes will follow the logistic growth model

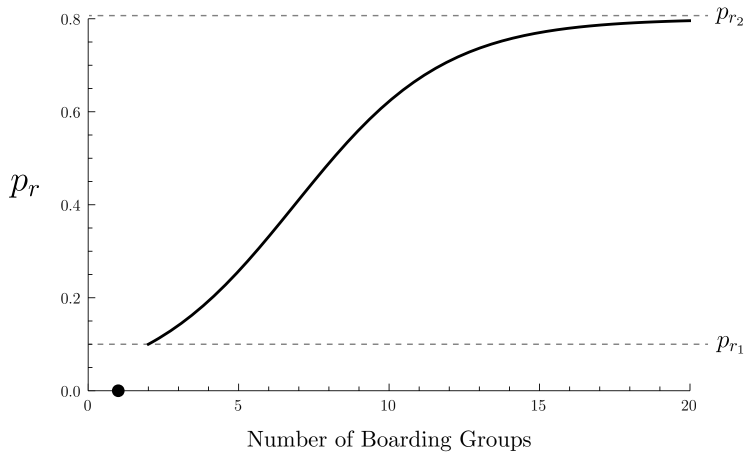


Figure 2: Reneging Rate ($p_{r_1} = 0.1$, $p_{r_2} = 0.8$, $R = 0.4$)

3.3 Classifying Passengers

In order to predict the behavior of passengers, we classify individual travelers based on individual boarding factors. These factors include the walking speed, standing/sitting speed, and the number of carry-on items. Within our model, these characteristics are stochastically generated for each passenger at the start of the simulation.

Walking Speed The rate at which passengers navigate aisles heavily influences the total boarding time. Given that an individual's age is the primary factor influencing such pace, we model walking speed accordingly. We consider the following average walking speeds of different age groups [7]:

Table 2: Walking speeds based on age [7]

Age Group	Walk Speed (m/s)
80 – 89	0.94 – 0.97
70 – 79	1.13 – 1.26
60 – 69	1.24 – 1.34
50 – 59	1.31 – 1.43
40 – 49	1.39 – 1.43
30 – 39	1.34 – 1.43
20 – 29	1.34 – 1.36
15 – 19	1.25 – 1.30
7 – 14	1.00 – 1.10
< 7	0.90 – 0.92

Note that we break our simulation down into discrete time intervals, and each unit of distance is equivalent to the width of the aisle (approximately 50 cm). We scale all speeds and distances accordingly based on the average width of airline aisles.

Sitting/Standing Speed Within our simulation, the time it takes a passenger to sit down or stand up depends heavily on their inherent walking speed. It additionally depends on the number of passengers obstructing their motion. They must walk approximately 2 seats forward while waiting for any passengers blocking their seat to walk approximately 2 seats worth of distance (to exit and return back to the seat). Hence, if a passenger reaches their seat, the time they take to sit down is modeled by:

$$t_s = k_{stand} \cdot \left(2w^{-1} + 2 \cdot |w'| \cdot \max\left\{(w'_1)^{-1}, (w'_2)^{-1}, ...\right\}\right) \tag{3.3.1}$$

Where w is the passenger's walking speed, w'_i is the ith obstructing passenger's walking speed, and $|w'|$ is the total number of obstructing passengers.

Luggage Every passenger may have at most 2 pieces of luggage as carry-ons. These items are stored in each row's overhead bin, which has a capacity for 4 total carry-ons. The time it takes one passenger to load their luggage into the carry-on container is directly related to their physical condition, and subsequently their walking speed. From the above intuition, it is reasonable to model the lifting of a carry-on to the approximate equivalent of walking 1 seat forward. In other words, walking 1 seat forward takes as much time as lifting one piece of luggage:

$$t_L = k_{luggage} \cdot w^{-1} \cdot \min\{c_L, L\} \tag{3.3.2}$$

Where w is the individual's walking speed, c_L is the current capacity of the overhead bin, and L is the number of luggage items the passenger is holding.

Some passengers may have more luggage than others— in fact, sometimes the capacity will not be sufficient for the entire row of passengers. In order to accommodate, rather than having passengers navigate around the plane to find open overhead bins, we have them hold onto and sit down with their carry-on. Once all passengers in the boarding group have sat down, the flight attendants walk around and place each piece of excess luggage into the closest available overhead bin. This incurs some minimal buffer time between boarding groups.

3.4 Passenger Demographics

When generating the set of passengers boarding an aircraft, the age distribution is not uniform (e.g., it is far more likely that a passenger is middle-aged than an infant). We model the passenger demographics as a Gaussian Distribution, which we sample from when generating individual passengers. This normal distribution is defined as follows:

$$D(a) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{a-\mu}{\sigma}\right)^2\right) \tag{3.4.1}$$

Where a is the passenger's age, μ is the mean age of airline passengers, and σ is the standard deviation. Based on the average American population, we set our passenger mean age $\mu = 40$ and standard deviation $\sigma = 10$ [9].

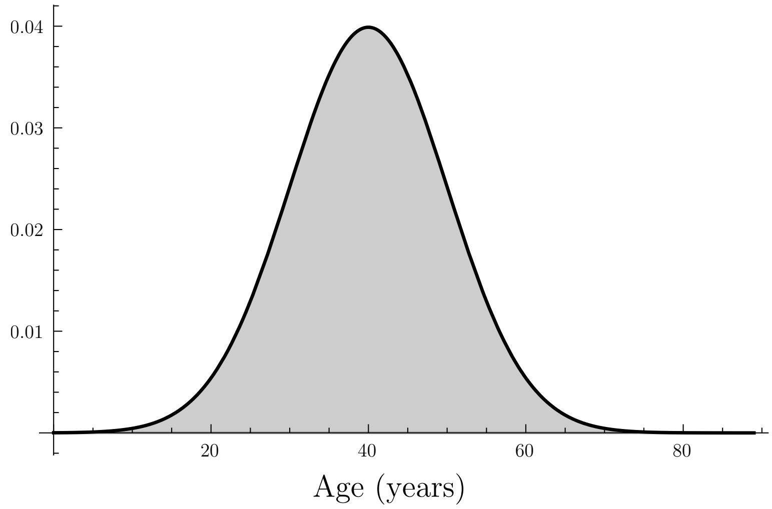


Figure 3: Distribution of Passenger Age ($\mu = 40$, $\sigma = 10$)

3.5 Structuring a Stochastic Process

In order to calculate the estimated boarding time of any procedure, we set up a stochastic process that simulates the boarding process. This approach allows for a flexible computation that matches the complex behavior of passengers. Repeating the simulation over a few thousand iterations, we are able to retrieve a distribution that reflects the strengths and weaknesses of a particular boarding procedure.

We design our simulation to consider a broader notion of an "aircraft": we break down the structure of airliners into directed acyclic graphs (DAR) with passengers traveling through aisles in one direction. Along aisle edges, we allow for the placement of "seat rows", which represent groups of passenger seats. These "seat rows" come equipped with designated overhead bins for passengers to utilize. Figure 4 demonstrates one such graph for a "Flying Wing" aircraft.

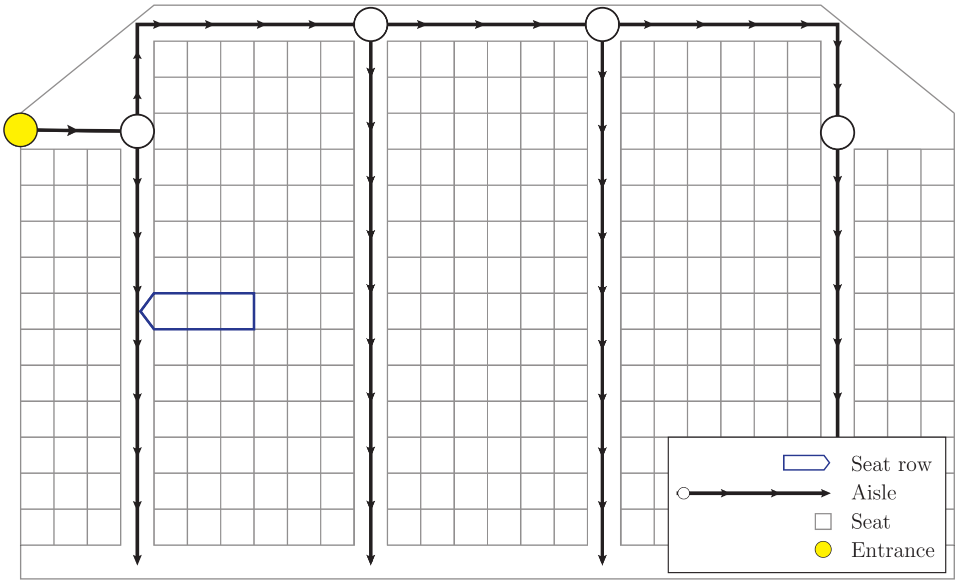


Figure 4: Unidirectional boarding graph of "Flying Wing" Aircraft

As passengers move through the aircraft, they behave according to their physical characteristics and boarding factors. For example, passengers are only able to move as fast as their walking speed allows; anyone walking slowly will hold up all passengers behind them. Additionally, when a passenger reaches their seat, they must stop and wait for any obstructions to clear. After they sit down, the rest of the line may continue boarding. As described in Section 3.3, passengers hold onto their carry-ons if the overhead bins fill up. After boarding finishes, flight attendants collect this excess luggage and sort it into the nearest available overhead bins. This buffer period occurs between group loading. We further describe the general structure of our simulation in Figure 5.

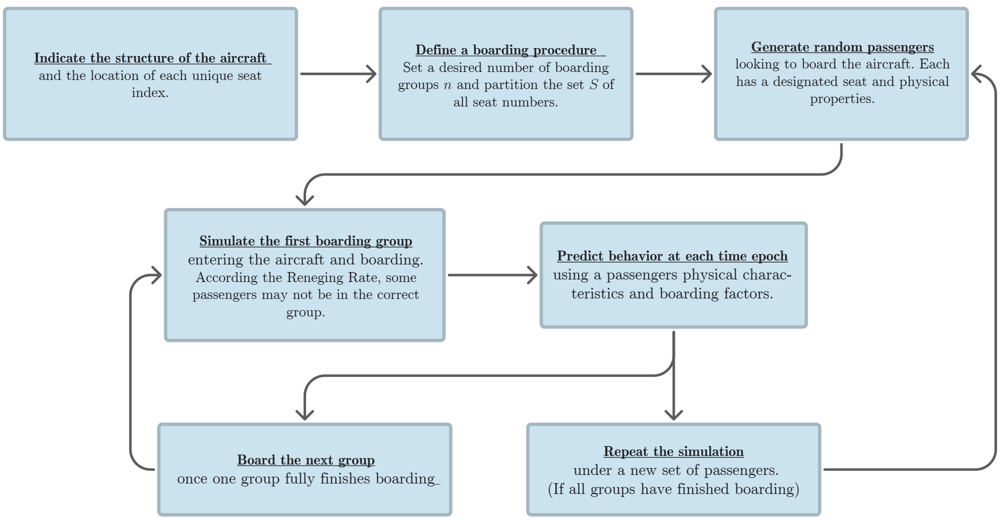


Figure 5: Flowchart of the boarding simulation model

3.6 Results & Analysis

We run our model on the "Narrow Body", "The Flying Wing", and "Two-Entrance, Two Aisle"[2] aircrafts, and consider three different boarding procedures for the "Narrow Body" aircraft. We consider the following pre-existing, and commonplace boarding procedures: *Random*, *Boarding by Section*, and *Boarding by Seat*.

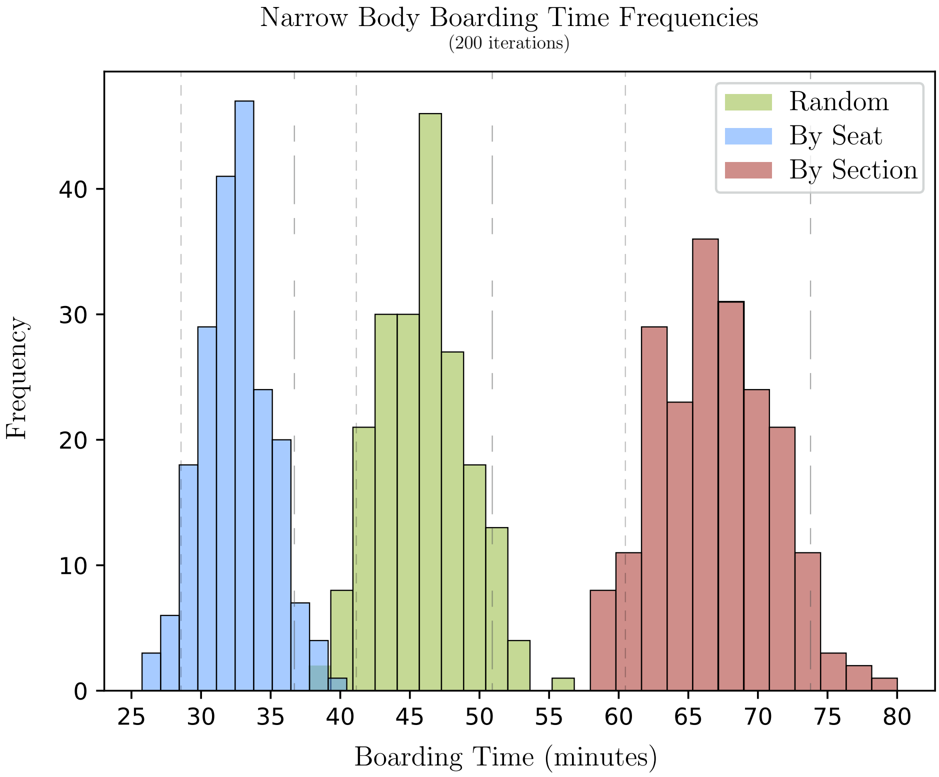


Figure 6: Distributions of Narrow Body boarding times

Figure 6 illustrates the distribution of practical boarding times under these procedures in the "Narrow Body" aircraft model. We in turn compute the practical maximum, average, and practical minimum boarding times in Table 3. It appears that the *By Seat* boarding procedure is the most efficient in terms of average and practical boarding time— this result matches intuition, as passengers are less likely to obstruct each other during the boarding process, as they sit down in order of the seat row. Of course the amount of reneging involved influences the magnitude of this boarding scheme's efficacy.

[2] As a simplifying assumption, we split this style of aircraft's boarding into two parts in order to maintain one direction of motion. This is reasonable since it is fair to assume passengers enter the aircraft from the side closest to their seat. This can be reasonably enforced by airport staff.

Table 3: Narrow Body boarding simulation ($k_{stand} = 50$, $k_{luggage} = 50$, $\mu = 40$, $\sigma = 10$, $R = 0.4$)

Procedure	groups	(p_{r_1}, p_{r_2})	Carry-ons	avg.	5^{th} percentile	95^{th} percentile
By Seat†	3	(0.1,0.8)	$0-2$	32.6 mins	28.6 mins	36.7 mins
Random†	1	(0.1,0.8)	$0-2$	46.0 mins	41.2 mins	50.9 mins
By Section†	3	(0.1,0.8)	$0-2$	66.9 mins	60.5 mins	73.8 mins
By Seat	3	(0.4,0.8)	$1-3$	62.3	57.24	69.8
Random	1	(0.4,0.8)	$1-3$	63.2	57.6	68.0
By Section	3	(0.5,0.9)	$1-3$	73.1	66.1	81.8

†*pictured in Figure 6*

In Table 3 we additionally perform a sensitivity analysis by varying the Reneging Rate and the number of average carry-ons: we fluctuate the values of p_{r_1}, p_{r_2}, and the range of passenger carry-ons. Under varying conditions, it seems the *Random* procedure will be least influenced by reneging, as the group of passengers is already fully randomized. For the other sections, it appears that the *By Section* boarding procedure will always be worse than the random procedure due to two observations: as we increase the Reneging Rate, the procedure will be near-identical to randomness, but split into 3 groups. This means boarding will match 1 random group, but will have buffer time in between, thus increasing total time. For this same reason, as we increase the reneging, the *By Seat* procedure seems to also grows similar to the *Random* procedure.

The above approaches towards modeling and simulating passenger behavior benefit from a few key strengths: mainly, the low number of input parameters that account for the major factors associated in the boarding process. Additionally, the stochastic and iterative nature of the simulation capture some of the chaos involved in such predictions.

While our approaches allow for a robust and adaptable model that can fit to many aircraft configurations, this model has its limitations. For example, our model does not consider familial or cohort passengers—groups that enter and leave the plane together. These groups may account for a large portion of total boarding and thus could be considered. Though such functionality is achievable within our approach by simply amending the behavior of specific passengers at each time epoch.

IV Boarding Optimization Model

In the following section, we consider the previously presented simulation model and consider autonomous methods of optimization. We present a set of near-optimal boarding procedures by pairing a genetic algorithm with our boarding simulation. We run this model on three classes of aircrafts: “Narrow Body”, “Flying Wing”, and “Two Entrance, Two Aisle”. We additionally provide analysis surrounding which boarding strategies work best.

4.1 Boarding Group Selection as an Evolutionary Process

Leveraging the simulation developed in Section III, we can start to form relationships between boarding group partitions and their corresponding boarding times, given a particular aircraft layout. Drawing concepts from Darwinian evolution and genetics, we formulate a partition as the genetic sequence of an organism and the corresponding boarding time as a heuristic estimate of the organism’s "fitness". By defining a population of randomly initialized partitions, we can replicate the process of natural selection to improve the overall fitness of the population over time. Figure 7 summarizes this selection process, and technical details are provided in Appendix A.

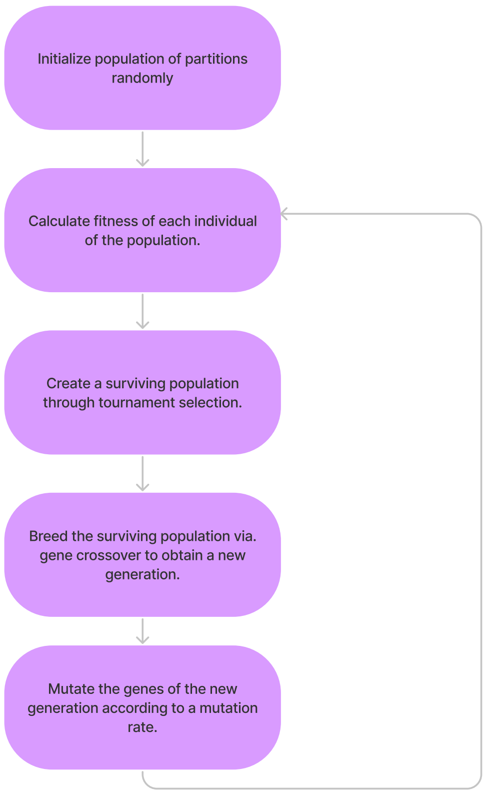


Figure 7: A flowchart of the genetic algorithm optimization procedure.

4.2 Results and Analysis

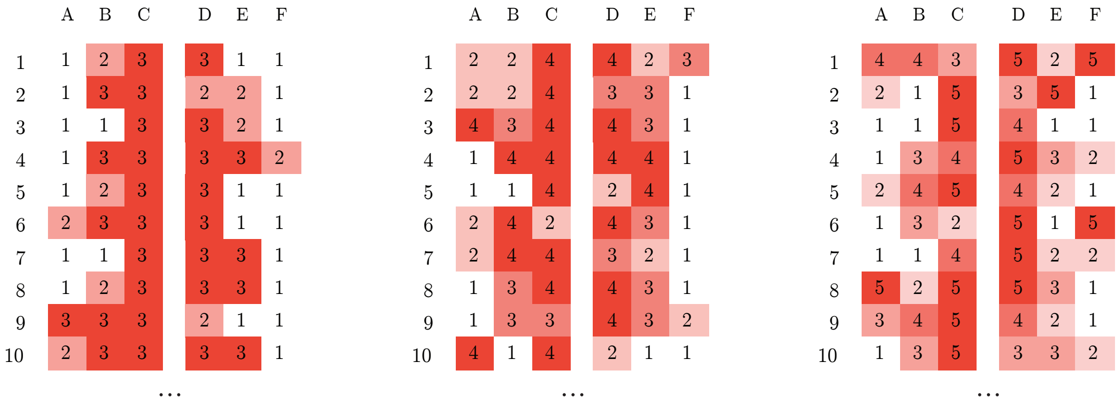


Figure 8: Optimal "Narrow Body" Aircraft Boarding Procedures boarding group sizes of 3,4, and 5

Figure 8 presents the results of this model under boarding group sizes of 3,4, and 5 for the "Narrow Body" aircraft. As can be seen, it appears that optimal boarding procedures involve strategies that are similar to those of the *By Seat* procedure— limiting the amount of interference within rows. In fact, it appears that under our simulated approach, the regular *By Seat* procedure (or slight alterations of it) is most efficient.

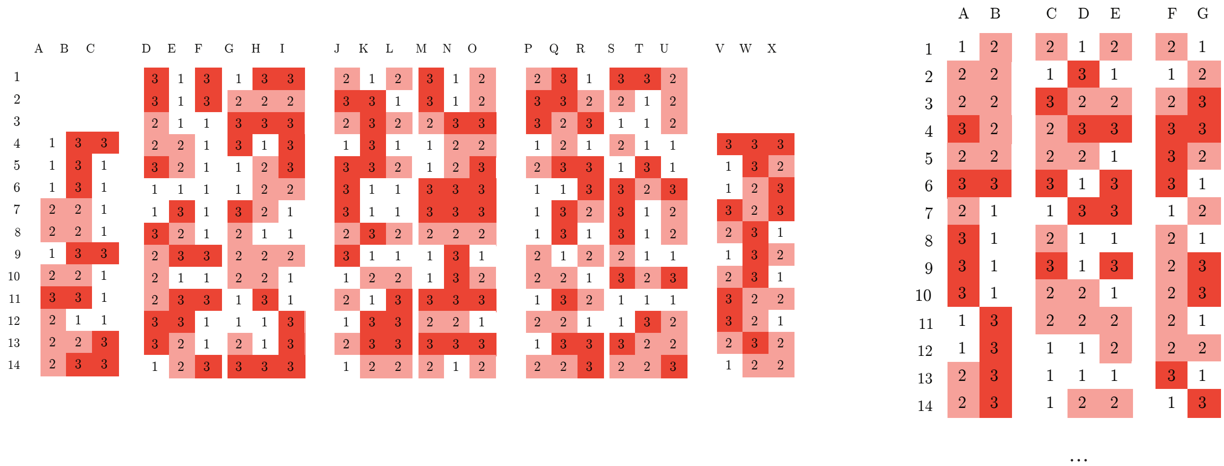


Figure 9: Optimal Aircraft Boarding Procedures for Flying Wing and Two Entrance, Two Aisle boarding group sizes of 3

This can further be seen in Figure 9, where we compare a set of optimal 3-group boarding procedures of both the "Flying Wing" and "Two Entrance, Two Aisle" aircraft. Much like the "Narrow Body", it appears the strategy of *By Seat* boarding persists. Naturally, there is noise within these results due to the stochastic nature of both the simulation and the genetic algorithm.

This approach captures complex relationships between similar boarding procedure partitions—

swapping seats amongst boarding groups can capture subtle improvements in configurations. Yet while this robust genetic algorithm has its limitations: mainly stemming from the stochastic nature of both the simulation and the optimization model itself. Subsequently, many near-optimal solutions can be found with minimal alterations (one or two seats swapped in the partition ordering).

4.2.1 Simulating and Optimizing Lower Capacity

Within both our simulations and our optimizations, we consider close to 100% passenger turn-up in order to provide sufficiently *general* procedures. If passenger turn-up is lower, as seen in pandemic situations where capacity is decreased, these models crucially **do not** fall apart. Our predictive model has an optional attendance parameter to indicate such capacity limitations, with limited effects on both predictive time trends and optimized partitions.

V Disembarking Models

Prior to considering the process of disembarking an aircraft, we make the following observations that may shed light on the enigmatic problem faced by airline companies. Firstly, disembarking an aircraft is inherently more difficult to predict and coordinate, as opposed to boarding. This is due to the impatience of passengers after a long flight, as well as the key notion of "ordered exiting"—passengers who sit closer to the aisles will naturally not allow other passengers in their row to leave prior to themselves. In other words, an individual sitting in the aisle seat will almost surely start disembarking if they are told to allow the window-seat passenger to pass—if they need to get up from their seat, they might as well leave as well.

With this key observation, along with the implicitly high reneging rate of passengers who just experienced a multi-hour flight, we consider the problem of disembarking. We propose two models and approaches toward this problem, though we emphasize that there is far less predictability in the deplaning process.

5.1 Model #1: A Simulated Approach

We note that predicting the total disembarking time can be achieved with the same approach as the boarding simulation. In fact, the same exact model may be run in reverse to compute disembarking time. The only practical differences will lie in the Reneging Rate, as passengers will disobey the ordering at a much higher rate. Thus, since the crowds will almost always mix and not follow the ordering, these simulations will likely appear near identical in terms of timing.

While we note that the practical time-saves involved with optimizing "Narrow Body" (or any aircraft for that matter) disembarking will not be as significant as boarding, there are notable approaches. With the simulated model, we use the above observation of "ordered exiting" to base our optimization model. The optimal general disembarking procedure will involve the *reverse* of the optimal boarding procedure, under the constraint that this boarding procedure sits passengers down *in order of seats*. In other words, we choose an optimal partition where all window passengers sit down prior to middle seated passengers, followed by aisle seats.

This immediately places the reverse of the *By Seat*[3] boarding as an optimal disembarking method— it fits the above criteria and benefits from a low number of disembarking groups, as well as being easy for passengers to understand. In the following section, we outline a more mathematical approach in seeing why this disembarking procedure is so effective.

[3] This boarding category additionally considers equivalent procedures within alternate aircrafts are, like the Flying Wing.

5.2 Model #2: Mathematical Maneuvering

5.2.1 Predicting Disembarking

A more generalized, formulaic justification for the above can be developed with a recursive approach. More concretely, a successful model that computes the total disembarking time for a commercial aircraft, simply needs to accurately compute the time needed for the last passenger to exit the aircraft. Following from our "ordered exiting" observation, this last passenger will most likely be the one in the window seat in the furthest back row. For an aircraft with j passengers with the jth one being furthest back, this passenger will have to wait:

$$t_j = t_r + t_d + t_L + t_{j-1} \tag{5.2.1}$$

Where t_{j-1} is the disembarking time of the passenger ahead of them, t_L is the passenger's luggage retrieval time, t_r and t_d are the times spent exiting the row and aisles, respectively.

We can model the time to traverse the row and aisle respectively as: $t_r = \frac{d_{row}}{w_{avg}}$ and $t_d = \frac{d_{aisle}}{w_{avg}}$.

The recursive formula for the jth passenger disembarking time could be expressed as a single sum over each passenger:

$$t_j = t_r + t_d + t_L + t_{j-1} = \sum_{i=1}^{j} \frac{d_{aisle_i}}{r_{aisle_i}} + \frac{d_{row_i}}{r_{row_i}} + t_{L_i} \tag{5.2.2}$$

Computing such a time is fairly straightforward for any airplane layout by simply computing the distances of each row and seat from the exits. Although it crucially does not consider congestion delays within the aisles, it is accurate when passengers do not interfere. While this is reasonable when considering the average optimal case, where passengers do not block each other, it is not quite as robust within practical scenarios. Ultimately, this approach provides a theoretical view and underlying intuition behind our simulated model approach.

5.2.2 Minimizing Boarding Time

As outlined within our preliminary analyses, there are inherent practical limitations to single seat-based disembarking: beyond the problem of "ordered exiting", children are separated from parents and cohorts do not remain together. For this reason, as we previously noted, *By Seat* methods are theoretically efficient, but may have some practical concerns. As we mentioned in Model #1, this is the predominant/optimal approach we recommend.

As an alternate approach, we offer a modified *3^{rd}-Row Algorithm*, which applies to "Narrow Body", "Flying Wing", and "Two Aisle, Two Entrance" aircrafts. This algorithm consider the principle of releasing boarding groups by every 3rd row. Releasing groups in this way is beneficial in three primary ways: (1) congestion is limited to a maximum of those in a row, which is supported by the (2) space created by increasing the distance between disembarking rows and (3) not separating key cohorts.

For "Narrow Body", this is relatively straightforward and follows the above description. The "Flying Wing" and "Two Aisle, Two Entrance" follow by splitting aisles into independent aircrafts. Within "Flying Wing", we can divide 5 cabins into four single aisle regions with row lengths of 3 seats. With each of these four single aisle regions, we treat these regions as if they were single aisle aircrafts. This means that we assume that passengers on these regions cannot traverse and unloading luggage from aisles in different regions. For "Two Aisle, Two Entrance", although there are two exits, there are also twice as many passengers. Due to the symmetric nature of the entrance-aisle configuration, it suffices to figure out how to optimize the disembarking time for one half of the seats and a single entrance, as the other half of seats and entrance would take the same amount of time.

Analogous to our boarding models, these approaches for disembarking will persist when the number of passengers is less than full capacity (under the condition that the passengers are uniformly distributed). Applying the 3rd-row algorithm to such a situation wouldn't not only optimize boarding times but do so while maintaining the same COVID distance boarding protocols. If the COVID distancing protocol is neglected for a quicker disembarking time, then the time can be optimized by alternating between a lower number of rows.

References

[1] Eitan Bachmat et al. "Analysis of Airplane Boarding Times". en. In: *Operations Research* 57.2 (Apr. 2009), pp. 499–513. ISSN: 0030-364X, 1526-5463. DOI: 10.1287/opre.1080.0630. URL: http://pubsonline.informs.org/doi/abs/10.1287/opre.1080.0630 (visited on 03/07/2022).

[2] Jason H. Steffen and Jon Hotchkiss. "Experimental test of airplane boarding methods". en. In: *Journal of Air Transport Management* 18.1 (Jan. 2012), pp. 64–67. ISSN: 0969-6997. DOI: 10.1016/j.jairtraman.2011.10.003. URL: https://www.sciencedirect.com/science/article/pii/S0969699711000986 (visited on 03/07/2022).

[3] Serter Iyigunlu, Clinton Fookes, and Prasad Yarlagadda. "Agent-based modelling of aircraft boarding methods". In: *2014 4th International Conference On Simulation And Modeling Methodologies, Technologies And Applications (SIMULTECH)*. Aug. 2014, pp. 148–154. DOI: 10.5220/0005033601480154.

[4] Rahul Bidanda et al. "A REVIEW OF OPTIMIZATION MODELS FOR BOARDING A COMMERCIAL AIRPLANE". In: Aug. 2017.

[5] Shafagh Jafer and Wei Mi. "Comparative Study of Aircraft Boarding Strategies Using Cellular Discrete Event Simulation". en. In: *Aerospace* 4.4 (Dec. 2017). Number: 4 Publisher: Multidisciplinary Digital Publishing Institute, p. 57. ISSN: 2226-4310. DOI: 10.3390/aerospace4040057. URL: https://www.mdpi.com/2226-4310/4/4/57 (visited on 03/07/2022).

[6] Tie-Qiao Tang et al. "An aircraft boarding model with the group behavior and the quantity of luggage". en. In: *Transportation Research Part C: Emerging Technologies* 93 (Aug. 2018), pp. 115–127. ISSN: 0968-090X. DOI: 10.1016/j.trc.2018.05.029. URL: https://www.sciencedirect.com/science/article/pii/S0968090X1830768X (visited on 03/07/2022).

[7] *Average Walking Speed: Pace, and Comparisons by Age and Sex*. en. Mar. 2019. URL: https://www.healthline.com/health/exercise-fitness/average-walking-speed.

[8] *Logistic function*. en. Page Version ID: 1076600991. Mar. 2022. URL: https://en.wikipedia.org/w/index.php?title=Logistic_function&oldid=1076600991 (visited on 03/12/2022).

[9] *Median Age by State 2022*. URL: https://worldpopulationreview.com/state-rankings/median-age-by-state.

[10] *Normal distribution - Wikipedia*. URL: https://en.wikipedia.org/wiki/Normal_distribution (visited on 03/12/2022).

VI Appendix A: Boarding Simulation Code

```python
from collections import OrderedDict
import numpy as np
import random
import math

# all distances/spacing are with respect to the width of the aisle/passenger
# e.g.:
#   1   = width of aisle
# 0.2   = 20% of the aisle width

# spacing between passengers as they walk
PERSONAL_SPACE   = 0.2
# reneging rate coefficients
MINIMUM_RENEGING     = 0.1 # With two rounds, reneging occurs at a rate of approximately ___
MAXIMUM_RENEGING     = 0.8 # As the number of rounds increases, reneging approaches a rate of ___
GROWTH_RATE_RENEGING = 0.4

# passenger walking speed data. aisle/passenger width = 50 cm = 0.5 m (approximately)
# in cm/(0.01 sec)
# 1 time unit is approx 0.1 seconds
WALKING_SPEEDS   = OrderedDict({
    7:  [0.090 / 0.5, 0.092 / 0.5],    # [estimate]    for ages 0-6      90- 92 cm / sec
    15: [0.100 / 0.5, 0.110 / 0.5],    # [estimate]    for ages 7-14    100-110 cm / sec
    20: [0.125 / 0.5, 0.130 / 0.5],    # [estimate]    for ages 15-19   125-130 cm / sec
    29: [0.134 / 0.5, 0.136 / 0.5],    # [cited data]  for ages 20-29   134-136 cm / sec
    39: [0.134 / 0.5, 0.143 / 0.5],    # [cited data]  for ages 30-39   134-143 cm / sec
    49: [0.139 / 0.5, 0.143 / 0.5],    # [cited data]  for ages 40-49   139-143 cm / sec
    59: [0.131 / 0.5, 0.143 / 0.5],    # [cited data]  for ages 50-59   131-143 cm / sec
    69: [0.124 / 0.5, 0.134 / 0.5],    # [cited data]  for ages 60-69   124-134 cm / sec
    79: [0.113 / 0.5, 0.126 / 0.5],    # [cited data]  for ages 70-79   113-126 cm / sec
    89: [0.094 / 0.5, 0.097 / 0.5],    # [cited data]  for ages 80-89    94- 97 cm / sec
})

# passenger age
MEAN_AGE = 40
STD_AGE  = 10

# sit/stand speed coefficients
STAND_COEFFICIENT = 50     # intuitively: Standing + sitting is approx ___x more taxing than stepping 50 cm forward
LUGGAGE_COEFFICIENT = 50    # intuitively: Lifting a carry-on is approx ___x more taxing than stepping 50 cm forward




class Passenger:

    def __init__(self, speed, carryons, seadId) -> None:
        self.walkingSpeed = speed
        self.carryons = carryons
        self.positionAlongPath = 0
        self.actionTimer = -2

        self.seatNumber = seadId
        self.seatLocation = None
        self.rowIndex = None
        self.sitting = False

    def __str__(self) -> str:
        return f"(#{self.seatNumber}, {self.positionAlongPath})"

    def setSeatLoc(self, loc, ind):
        self.seatLocation = loc
        self.rowIndex = ind

    def walk(self, infront=None, d=1):
        # dont walk past seat or passenger infront
        newPosition = self.positionAlongPath + self.walkingSpeed
        if infront:
            if (infront.getPosition() - newPosition) < (PERSONAL_SPACE + 1):
                newPosition = infront.getPosition() - PERSONAL_SPACE - 1
        self.positionAlongPath = min(newPosition, self.seatLocation - d)

    def resetPosition(self):
        self.positionAlongPath = 0

    def getSpeed(self):
        return self.walkingSpeed

    def getNumber(self):
        return self.seatNumber

    def getPosition(self):
        return self.positionAlongPath

    def getLoc(self):
        return self.seatLocation
```

```python

    def getRowIndex(self):
        return self.rowIndex

    def readyToSit(self, d=1):
        distanceFromSeat = (self.positionAlongPath - self.seatLocation)
        if distanceFromSeat >= self.walkingSpeed:
            raise Exception("Passenger has walked past their seat")
        return distanceFromSeat >= (0-d)

    def getTimer(self):
        return self.actionTimer

    def setSitting(self):
        self.sitting = True

    def isSitting(self):
        return self.sitting

    def updateTimer(self):
        self.actionTimer -= 1

    def setTimer(self, val):
        self.actionTimer = val

    def getCarryons(self):
        return self.carryons



class Aisle:
    def __init__(self, l):
        self.length = l
        self.seats = {}
        self.queue = []

    def __str__(self) -> str:
        return f"Aisle of length {self.length} \n With the following seat positions: {[[str(j) for j in i] for i in self.seats.values()]} \n And queue: {[str(p) for p in self.queue]}"

    def getQueue(self):
        return self.queue

    def isSeatInAisle(self, passenger):
        for pos in self.seats.keys():
            for row in self.seats[pos]:
                if row.checkSeat(passenger.getNumber()):
                    return True
        return False

    def isAisleClear(self):
        if len(self.queue) < 1:
            return True
        return self.queue[0].getPosition() >= 1

    def addToQueue(self, passenger):
        for pos in self.seats.keys():
            for i, row in enumerate(self.seats[pos]):
                if row.checkSeat(passenger.getNumber()):
                    self.queue.append(passenger)
                    self.queue[-1].setSeatLoc(pos, i)
                    self.queue[-1].resetPosition()
                    return
        raise Exception("Passenger's row is not in the aisle!")

    def popQueue(self):
        return self.queue.pop()

    def addRow(self, position, row):
        if not position in self.seats.keys():
            self.seats[position] = []
        self.seats[position].append(row)

    def movePassengers(self):
        for i, passenger in enumerate(self.queue):
            nearbyRows = self.seats[passenger.getLoc()]
            # check there is room and the passenger is in position to sit down
            if passenger.getTimer() > -2:
                if passenger.getTimer() <= 0:
                    nearbyRows[passenger.getRowIndex()].seatPassenger(passenger)
                    nearbyRows[passenger.getRowIndex()].stoweAway(passenger.getCarryons())
                    self.queue[i].setSitting()

                else:
                    self.queue[i].updateTimer()

            elif passenger.readyToSit():
                # passenger sits down
                waitTime = (2 / passenger.getSpeed()) + nearbyRows[passenger.getRowIndex()].obstructionTime(passenger.getNumber())
                waitTime += LUGGAGE_COEFFICIENT * passenger.getCarryons() * (1 / passenger.getSpeed())
                self.queue[i].setTimer(math.ceil(waitTime))
            else:
```

```python
                self.queue[i].walk(infront=self.queue[i-1] if i > 0 else None,d=1)

        self.queue = list(filter(lambda p : not p.isSitting(), self.queue))

    def doneBoarding(self):
        return len(self.queue) < 1


class SeatRow:
    def __init__(self, seatIds):
        self.row = OrderedDict()
        for id in seatIds:
            self.row[id] = None
        self.overheadCapacity = 4

    def __str__(self) -> str:
        return ' '.join([str(i) for i in self.row.values()])

    def seatPassenger(self, passenger):
        if not passenger.getNumber() in self.row.keys():
            raise Exception("Passenger attempting to sit in wrong row")
        self.row[passenger.getNumber()] = passenger

    def checkSeat(self, id):
        return id in self.row.keys()

    def obstructionTime(self, id):
        if not (id in self.row.keys()):
            raise Exception('Passenger is not in this row!')
        obstructors = [1 / self.row[i].getSpeed() for i in self.row.keys() if (self.row[i] and (list(self.row.keys()).index(id) <
            list(self.row.keys()).index(i)))]
        return 2 * STAND_COEFFICIENT * len(obstructors) * max(obstructors, default=0)

    def numPassCurrentlyInRow(self):
        return len([i for i in self.row.values() if i])

    def stoweAway(self, toStore):
        self.overheadCapacity -= toStore

    def getAvailableOverhead(self):
        return self.overheadCapacity

class MainAisle:
    def __init__(self, lengthAisle) -> None:
        self.length = lengthAisle
        self.aisles = {}
        self.queue = []

    def addAisle(self, aisleLoc, toAdd):
        if aisleLoc in self.aisles.keys():
            raise Exception('There is already an aisle placed there!')
        self.aisles[aisleLoc] = toAdd

    def addBoardingGroup(self, boardingGroup):
        self.queue = boardingGroup
        for i, p in enumerate(self.queue):
            for j in self.aisles.keys():
                if self.aisles[j].isSeatInAisle(p):
                    self.queue[i].setSeatLoc(j, None)

        for p in self.queue:
            if p.getLoc():
                continue
            raise Exception("A passenger cant find an aisle. Their seat is not in any aisle!", str(p))

    def update(self):
        toRemove = []
        for i, p in enumerate(self.queue):
            if p.readyToSit(0.1):
                if self.aisles[p.getLoc()].isAisleClear():
                    toRemove.append(i)
            else:
                self.queue[i].walk(infront=self.queue[i-1] if i > 0 else None,d=0)

        for i in self.aisles.keys():
            self.aisles[i].movePassengers()

        self.queue = [p for ind, p in enumerate(self.queue) if not (ind in toRemove)]

    def isFinishedBoarding(self):
        if len(self.queue) > 0:
            return False
        for aisle in self.aisles.values():
            if len(aisle.getQueue()) > 0:
                return False
        return True

    def __str__(self) -> str:
        return ' '.join([str(i) for i in self.queue]) + "\n" + '\n'.join([str(i) for i in self.aisles.values()])


def GenerateRandomPassenger(id):
```

```python

    age = np.random.normal(MEAN_AGE, STD_AGE, 1)[0]
    speed = 1
    for a in list(WALKING_SPEEDS.keys()):
        if age < a:
            speed = np.random.uniform(WALKING_SPEEDS[a][0], WALKING_SPEEDS[a][1], 1)[0]
            break

    carry = random.choice([0,1,2])
    return Passenger(speed, carry, id)

def renegingRate(numGroups):
    if numGroups == 1:
        return 0
    return (MINIMUM_RENEGING*MAXIMUM_RENEGING*math.exp(GROWTH_RATE_RENEGING*(numGroups - 2)) / (MAXIMUM_RENEGING +
    ↪   MINIMUM_RENEGING*(math.exp(GROWTH_RATE_RENEGING*(numGroups - 2))-1)))
```

```python
from cProfile import run
from boardingSim import *
import random
import matplotlib.pyplot as plt
import pickle
import numpy as np


# seats are consecutively placed
SEAT_WIDTH = 1

# partition is a 2d array of all seat indices from 1...numPassengers
# attendance is the approximate % of seats filled up (how many passengers
↪   showed up to board the aircraft)
# numPassengers is the total number of passengers that can fit on the aircraft
↪   (the total # of seats)
ROWS = 10

def runSim(partition, attendance=1, numPassengers=ROWS*6):
    NARROW_BODY = Aisle(35)
    c = 1
    # Initialize the aircraft
    for i in range(ROWS):
        NARROW_BODY.addRow(2+i*(SEAT_WIDTH), SeatRow([c, c+1, c+2]))
        c += 3
        # TODO: order here matters (not an actual todo, but should remove this
        ↪   comment later)
        NARROW_BODY.addRow(2+i*(SEAT_WIDTH), SeatRow([c+2, c+1, c]))
        c += 3

    capacity = c-1
    totalTime = 0

    boardingGroups = [[] for subset in partition]

    for index in range(1, numPassengers+1):
        # probability a passenger showed up
        if random.random() < attendance:
```

```
            assignedGroup = 0
            nonAssignedGroup = []
            for groupidx, group in enumerate(partition):
                if index in group:
                    assignedGroup = groupidx
                    continue
                nonAssignedGroup.append(groupidx)

            # decide whether the passenger reneges or stays in the assigned
            ↪  group
            if (len(boardingGroups) > 1) and (random.random() <
            ↪  (renegingRate(len(boardingGroups)))):
                boardingGroups[random.choice(nonAssignedGroup)].append(index)
            else:
                boardingGroups[assignedGroup].append(index)

    # each boarding group enters in a random order
    for group in range(len(boardingGroups)):
        random.shuffle(boardingGroups[group])

    # print(boardingGroups)
    for group in boardingGroups:
        for index in group:
            NARROW_BODY.addToQueue(GenerateRandomPassenger(index))

        t = 0
        while not NARROW_BODY.doneBoarding():
            t += 1
            NARROW_BODY.movePassengers()
            # print(t/10, ','.join(str(p) for p in NARROW_BODY.getQueue()))
        # print(t)
        totalTime += t

    # print(NARROW_BODY)
    return totalTime
```

```
from boardingSim import *
import random
import matplotlib.pyplot as plt
import pickle
import numpy as np
import copy

FLYING_WING = MainAisle(29)
aisle5 = Aisle(14)
aisle26 = Aisle(14)
aisle12 = Aisle(14)
```

```
aisle19 = Aisle(14)
for i in range(11):
    ind = (i * 24)
    aisle5.addRow(i+1, SeatRow([1+ind, 2+ind, 3+ind]))
    aisle5.addRow(i+1, SeatRow([6+ind, 5+ind, 4+ind]))

    aisle12.addRow(i+1, SeatRow([7+ind,8+ind,9+ind]))
    aisle12.addRow(i+1, SeatRow([12+ind,11+ind,10+ind]))

    aisle19.addRow(i+1, SeatRow([13+ind,14+ind,15+ind]))
    aisle19.addRow(i+1, SeatRow([18+ind,17+ind,16+ind]))

    aisle26.addRow(i+1, SeatRow([19 + ind, 20 + ind, 21 + ind]))
    aisle26.addRow(i+1, SeatRow([24 + ind, 23 + ind, 22 + ind]))

for i in range(3):
    ind = (i * 18) + (24*11)

    aisle5.addRow(i+12, SeatRow([3+ind, 2+ ind, 1+ ind]))
    ind += 3
    aisle12.addRow(i+12, SeatRow([1+ind,2+ind,3+ind]))
    aisle12.addRow(i+12, SeatRow([6+ind,5+ind,4+ind]))

    aisle19.addRow(i+12, SeatRow([7+ind,8+ind,9+ind]))
    aisle19.addRow(i+12, SeatRow([12+ind,11+ind,10+ind]))
    ind += 12
    aisle26.addRow(i+12, SeatRow([1+ind,2+ind,3+ind]))

FLYING_WING.addAisle(5, aisle5)
FLYING_WING.addAisle(12, aisle12)
FLYING_WING.addAisle(19, aisle19)
FLYING_WING.addAisle(26, aisle26)


def runSim(partition, attendance=1, numpassengers=318):
    plane = copy.deepcopy(FLYING_WING)

    boardingGroups = [[] for subset in partition]

    for index in range(1, 318+1):
        # probability a passenger showed up
        if random.random() < attendance:
            assignedGroup = 0
            nonAssignedGroup = []
            for groupidx, group in enumerate(partition):
                if index in group:
                    assignedGroup = groupidx
```

```
                    continue
                nonAssignedGroup.append(groupidx)

            # decide whether the passenger reneges or stays in the assigned group
            if (len(boardingGroups) > 1) and (random.random() < (renegingRate(len(boardingGroups)))):
                boardingGroups[random.choice(nonAssignedGroup)].append(index)
            else:
                boardingGroups[assignedGroup].append(index)

    for group in range(len(boardingGroups)):
        random.shuffle(boardingGroups[group])

    totalTime = 0


    for group in boardingGroups:
        plane.addBoardingGroup([GenerateRandomPassenger(index) for index in group])

        t = 0
        while not plane.isFinishedBoarding():
            t += 1
            plane.update()
        totalTime += t

    return totalTime
```

```
from tkinter.tix import ROW
from boardingSim import *
import random
import matplotlib.pyplot as plt
import pickle
import numpy as np
import copy

ROWS = 14

# 2,..,10
TWO_AISLE = MainAisle(10)
aisle5 = Aisle(ROWS)
aisle9 = Aisle(ROWS)

c = 1

for r in range(1,1+ROWS):
```

```python
        aisle5.addRow(r, SeatRow([c, c+1]))
        c += 2
        d = 0
        if (r%2) == 0:
            aisle5.addRow(r, SeatRow([c+1, c]))
            d+=2
        else:
            aisle5.addRow(r, SeatRow([c]))
            d+=1
        c += d
        d = 0
        if (r%2) == 0:
            aisle9.addRow(r, SeatRow([c]))
            d+=1
        else:
            aisle9.addRow(r, SeatRow([c, c+1]))
            d+=2
        c += d
        aisle9.addRow(r, SeatRow([c+1, c]))
        c+=2
    TWO_AISLE.addAisle(5, aisle5)
    TWO_AISLE.addAisle(9, aisle9)





def runSim(partition, attendance=1, numpassengers=ROWS*7):
    plane = copy.deepcopy(TWO_AISLE)

    boardingGroups = [[] for subset in partition]

    for index in range(1, numpassengers+1):
        # probability a passenger showed up
        if random.random() < attendance:
            assignedGroup = 0
            nonAssignedGroup = []
            for groupidx, group in enumerate(partition):
                # group = [k+1 for k in g]
                if index in group:
                    assignedGroup = groupidx
                    continue
                nonAssignedGroup.append(groupidx)

            # decide whether the passenger reneges or stays in the assigned
            ↪ group
```

```
            if (len(boardingGroups) > 1) and (random.random() <
            ↪   (renegingRate(len(boardingGroups)))):
                boardingGroups[random.choice(nonAssignedGroup)].append(index)
            else:
                boardingGroups[assignedGroup].append(index)

    for group in range(len(boardingGroups)):
        random.shuffle(boardingGroups[group])

    totalTime = 0


    for group in boardingGroups:
        plane.addBoardingGroup([GenerateRandomPassenger(index) for index in
        ↪   group])

        t = 0
        while not plane.isFinishedBoarding():
            t += 1
            plane.update()
        totalTime += t

    return totalTime
```

VII Appendix B: Genetic Algorithm Code

```
import pandas as pd
from numpy.random import randint, choice
from narrowBody import runSim
from tqdm import tqdm
import pickle


class GA:
    def __init__(self, groups, population_size=100, attendance=1, n=198,
    ↪   mu=0.1, max_iter=100):
        self.groups = groups
        self.population_size = population_size
        self.attendance = attendance
        self.n = n
        self.mu = mu
        self.max_iter = max_iter
        self.population = pd.Series(dtype="object")

```

```python
    def reset(self):
        self.population = pd.Series(randint(0, self.groups, [self.population_size, self.n]).tolist())

    def fit(self, part):
        return runSim(part, self.attendance, self.n)

    def get_part(self, arr):
        out = [[] for _ in range(self.groups)]
        for i, v in enumerate(arr):
            # noinspection PyTypeChecker
            out[v].append(i + 1)
        return out

    def fit_population(self):
        return pd.Series([self.fit(self.get_part(arr)) for arr in self.population])

    def select(self, k=5):
        fit = self.fit_population()
        indices = [fit.sample(k).idxmin() for _ in range(self.population_size)]
        global_min = self.population.iloc[fit.idxmin()]
        self.population = pd.Series([self.population.iloc[i] for i in indices])
        return global_min, fit.min()

    def step(self):
        children = []
        for p1, p2 in zip(self.population.iloc[:-1:2], self.population.iloc[1::2]):
            c1, c2 = self.uniform_crossover(p1, p2)
            children.extend([self.mutate(list(c1), self.mu), self.mutate(list(c2), self.mu)])
        return pd.Series(children)

    def uniform_crossover(self, p1, p2):
        p = [p1, p2]
        a = randint(0, 2, len(p1))
        b = (~a.astype(bool)).astype(int)
        return zip(*[(p[v_a][i], p[v_b][i]) for i, (v_a, v_b) in enumerate(zip(a, b))])

    def mutate(self, x, mu):
        for i in range(len(x)):
            if choice([True, False], p=[mu, 1-mu]):
                x[i] = randint(0, self.groups)
        return x

```

```
    def run(self):
        memory = []
        self.reset()
        for _ in tqdm(range(self.max_iter)):
            global_min = self.select()
            memory.append(global_min)
            children = self.step()
            self.population = children
        return memory


if __name__ == '__main__':
    ga = GA(5)
    mem = ga.run()
    with open('save', 'wb') as f:
        pickle.dump(mem, f)
```