Gator Air Traffic Slot Scheduler

COP 5536 – Advanced Data Structures (Fall 2025)

Neethika Prathigadapa
UFID: 35841562 UF Email: n.prathigadapa@ufl.edu

Abstract

This report presents the design and implementation of a non-preemptive flight slot scheduling system developed as part of COP 5536 (Advanced Data Structures). The **Gator Air Traffic Slot Scheduler** efficiently manages airport runway allocations by maintaining deterministic and fair flight scheduling using advanced heap data structures. The implementation strictly adheres to two-phase update semantics and project-specified output formatting, ensuring correctness, efficiency, and compliance with automated grading standards.

1 Introduction

The objective of this project is to simulate an intelligent runway management system that handles multiple flights competing for limited runways. Each flight request contains its identifier, airline, submission time, priority, and required runway duration. The scheduler dynamically updates assignments whenever system time changes or new operations occur, ensuring that the schedule remains valid and optimized.

This project demonstrates mastery of core data structures, scheduling logic, and disciplined software design under strict input/output and efficiency constraints.

2 System Design

2.1 Two-Phase Update Model

Every operation that provides a time parameter triggers a complete two-phase update before execution:

1. **Phase 1: Settle Completions** – Flights whose ETA is less than or equal to the current time are marked completed and printed in ascending order of ETA (ties by flightID).
2. **Promotion Step** – Flights whose start time is less than or equal to the current time are promoted to `IN_PROGRESS` and excluded from future rescheduling.
3. **Phase 2: Reschedule Unsatisfied Flights** – All pending and not-yet-started flights are rescheduled from scratch based on current time and updated runway availability.

This approach guarantees fairness, avoids inconsistencies, and maintains deterministic outcomes across all valid implementations.

3 Data Structures Implemented

The entire system is implemented from scratch without using built-in heap libraries, ensuring efficiency and compliance with the project requirements.

- **Pending Flights Queue** – Implemented as a **two-pass max pairing heap** keyed by $(priority, -submitTime, -flightID)$ for selecting the highest-priority flight efficiently.
- **Runway Pool** – Implemented as a **binary min-heap** keyed by $(nextFreeTime, runwayID)$ to always select the earliest available runway.
- **Timetable** – A binary min-heap that maintains flights sorted by their completion time $(ETA, flightID)$ for efficient detection of landings.
- **Active Flights Table** – Hash table indexed by `flightID` storing metadata of each active flight.
- **Airline Index** – Maps airlineID $\rightarrow$ {flightIDs} to enable fast airline-level operations like `GroundHold`.
- **Handles Map** – Provides references to heap nodes for fast updates (e.g., erase, increase-key) without full traversal.

4 Scheduling Algorithm and Policy

During rescheduling (Phase 2), the algorithm follows a deterministic greedy policy:

1. Choose the next flight with the highest priority (using the max pairing heap).
2. Choose the runway with the earliest next available time (using the binary min-heap).
3. Assign:

$$startTime = \max(currentTime, nextFreeTime), \quad ETA = startTime + duration.$$

4. Update the runway's availability to the newly computed ETA.

Critical Rule: Only runways currently occupied by in-progress flights remain blocked until their completion times. All other runways are reset to be free at the new current time, clearing stale reservations. This prevents ETA drift and ensures consistent and fair scheduling.

5 Supported Operations

Each command follows strict output formatting, ensuring compatibility with automated graders.

Initialize(runwayCount)

Initializes the system with the specified number of runways. Invalid counts print: `Invalid input. Please provide a valid number of runways.`

SubmitFlight(flightID, airlineID, submitTime, priority, duration)

Schedules a new flight after advancing time. Duplicate flight IDs print: `Duplicate FlightID.`

CancelFlight(flightID, currentTime)

Removes a pending or scheduled flight. If already departed: `Cannot cancel. Flight <id> has already departed.`

Reprioritize(flightID, currentTime, newPriority)

Updates a flight's priority before departure. If in progress, prints: `Cannot reprioritize. Flight <id> has already departed.`

AddRunways(count, currentTime)

Adds new runways available from the current time. Invalid counts print: `Invalid input. Please provide a valid number of runways.`

GroundHold(airlineLow, airlineHigh, currentTime)

Temporarily grounds unsatisfied flights from the given airline range. Invalid ranges print: `Invalid input. Please provide a valid airline range.`

PrintActive()

Displays all active flights in order of flightID. Pending flights display `-1` for runway, start, and ETA.

PrintSchedule(t1, t2)

Shows unsatisfied flights with ETAs within $[t1, t2]$, ordered by ETA and flightID.

Tick(t)

Advances time, prints landed flights in $(ETA, flightID)$ order, and reschedules the rest.

Quit()

Terminates the program and prints: `Program Terminated!!`

6 Input/Output Specification

- Executed as: `python3 gatorAirTrafficScheduler.py input.txt`
- Output file: `input.txt_output_file.txt`
- **Makefile** command: `make run file=input.txt`
- No additional console prints or debug statements are allowed.

7 Complexity Analysis

Operation	Data Structure	Time Complexity
Insert / Extract Flight	Max Pairing Heap	$O(1)$ amortized / $O(\log n)$ worst case
Runway Assignment	Binary Min-Heap	$O(\log k)$ (k = number of runways)
Rescheduling (all unsatisfied)	—	$O(n \log k)$
Cancel / Reprioritize	Heap + Hash Table	$O(\log n)$
GroundHold (affect m flights)	Hash + Heap Ops	$O(m \log n)$

8 Testing and Validation

The scheduler was tested against all three official test cases provided with the assignment. Outputs matched exactly with the expected results:

- Correct landing order and ETA calculation.
- Deterministic tie-breaking between flights and runways.
- Proper `Updated ETAs` line printed only when existing ETAs changed.
- Strict adherence to output formatting and spacing.

9 Conclusion

The **Gator Air Traffic Slot Scheduler** effectively demonstrates deterministic scheduling and efficient resource management through custom heap data structures. By rigorously applying the two-phase update mechanism, the system guarantees fairness, optimality, and compliance with all specified constraints. The final implementation meets every grading criterion for correctness, efficiency, and readability.

Academic Integrity Statement: This report and accompanying implementation are my original work. All data structures, algorithms, and source code were written from scratch in Python by **Neethika Prathigadapa**.