Parcel Delivery Optimizer

Course Title: Computer Algorithms Lab
Course Code: CSE-2422
Session: Spring-2025
Semester: 4th

Submitted By:
Tahasina Tasnim Afra (C233456)
Asliraf Samaylan (C233446)
Nusrat Jahan (C233464)

Instructor:
Ms. Miskatul Jannat Tuly
Assistant Lecturer
Department of Computer Science and Engineering
International Islamic University Chittagong

Submission Date: August 7, 2025

Contents

List of Figures

1 Abstract

The Parcel Delivery Optimizer is a C++ console application built in Code::Blocks to streamline parcel delivery management. It supports Admin, Staff, and User roles for adding, assigning, tracking, and canceling parcels. Using Merge Sort, Greedy, and Knapsack algorithms, it optimizes delivery schedules and resource use. Data is stored in staff.txt and parcels.txt in a fixed-width tabular format, ensuring readability for non-technical users. This project showcases practical programming and algorithm design for logistics.

2 Introduction

The Parcel Delivery Optimizer is a C++ console application designed to efficiently manage parcel delivery operations. Developed using Code::Blocks, it enables users to add, track, assign, and optimize parcel deliveries within a logistics system. The program supports three user roles: Admin (manages parcels and staff), Delivery Staff (updates parcel status), and User (tracks or cancels parcels). It utilizes data structures such as vectors and structs to store parcel and staff information, and implements algorithms including Merge Sort, Greedy Scheduling, and Knapsack to optimize delivery processes. Data is stored in staff.txt and parcels.txt in a fixed-width tabular format, ensuring easy readability in text editors or spreadsheet software. This project demonstrates the practical application of programming concepts, algorithms, and file handling to address real-world logistics challenges in a user-friendly and organized manner.

3 Objective

The objectives of the Parcel Delivery Optimizer are:

- Develop a C++ console-based system using Code::Blocks to manage parcel delivery operations for Admins, Delivery Staff, and Users.
- Provide role-based functionality: Admins to manage parcels and staff, Staff to update parcel status and location, and Users to track or cancel parcels.
- Implement optimization algorithms—Merge Sort (for distance sorting), Greedy (for delivery time and staff assignment), and Knapsack (for weight optimization).
- Store data persistently in staff.txt and parcels.txt using a fixed-width tabular format, ensuring readability for non-technical users in text editors or spreadsheets.
- Demonstrate practical applications of data structures and algorithms.

4 Requirement Analysis

4.1 Hardware Requirements

- **Processor**: Minimum 1.5 GHz CPU (e.g., Intel Core i3 or equivalent).
- **RAM**: Minimum 4 GB (recommended 8 GB).
- **Storage**: Minimum 1 MB free disk space (may increase with data).
- **Input/Output**: Keyboard and monitor with standard terminal support.

4.2 Software Requirements

- **Operating System**: Windows, macOS, or Linux (with compatible compiler).
- **Development Environment**: Code::Blocks IDE with MinGW, or other IDEs (e.g., Visual Studio, Eclipse) with C++11 compiler.
- **Libraries**: Standard C++11 libraries (e.g., `iostream`, `vector`, `fstream`).
- **Optional**: Text editor (e.g., Notepad++) or spreadsheet software (e.g., Microsoft Excel) for data file viewing.

5 System Design

The Parcel Delivery Optimizer is a modular C++ console application developed in Code::Blocks to manage parcel delivery operations efficiently. Its key components are:

- **Data Structures**:
  - `Parcel` struct: Stores details such as `id` , `sender`, `receiver`, `status` (e.g., Booked, Delivered), `deliveryTime` , `distance` (km), `weight` (kg), `location`, `assigned` (boolean), and `assignedStaffId` (staff ID or -1 if unassigned).
  - `Staff` struct: Stores `id`, `name`, `joiningDate`, and `totalAssigned` parcels.
  - `Vectors`: `parcels` (all parcels) and `staffList` (all staff members).
  - Global variable `nextParcelId` for generating parcel IDs.
- **File Handling**: Data is stored in `staff.txt` and `parcels.txt` using a fixed-width tabular format with headers (e.g., `ID`, `SENDER`, `STATUS`) and separators (`+----...+`) for easy readability in both text editors and spreadsheets. Functions like `saveStaffData`, `loadStaffData`, `saveParcelData`, and `loadParcelData` handle persistence.
- **Algorithms**:
  - **Merge Sort**: Sorts parcels by distance for route optimization ($O(n \log n)$).
  - **Greedy Scheduling**: Includes -
    - EDF (Earliest Deadline First): Orders parcels by earliest delivery deadline.
    - Round Robin: Assigns parcels to staff in a balanced, cyclic manner.

    Overall complexity $O(n \log n)$.
  - **0/1 Knapsack**: Maximizes deliverable weight within a 10 kg capacity using dynamic programming ($O(n \times W)$).
- **User Interface**: Console menus use centered text (`centerText`) and dashed separators (`printLine`) for clarity, with role-specific options for Admin, Staff, and User.
- **Workflow**: At startup, data is loaded from files. Users perform role-based operations (e.g., add parcels, assign staff, track deliveries), and updates are saved automatically after changes or on exit.

6 Project Flow Diagram

The Parcel Delivery Optimizer program starts by loading data from `staff.txt` and `parcels.txt`, then offers role-based operations via a central menu. Admin manages parcels and staff with optimization, Staff updates statuses, and User tracks or cancels parcels, with data saved on exit.

A flowchart, shown in Figure 1 below, visualizes this flow, created in Lucidchart to highlight the menu's role-based branching and termination.

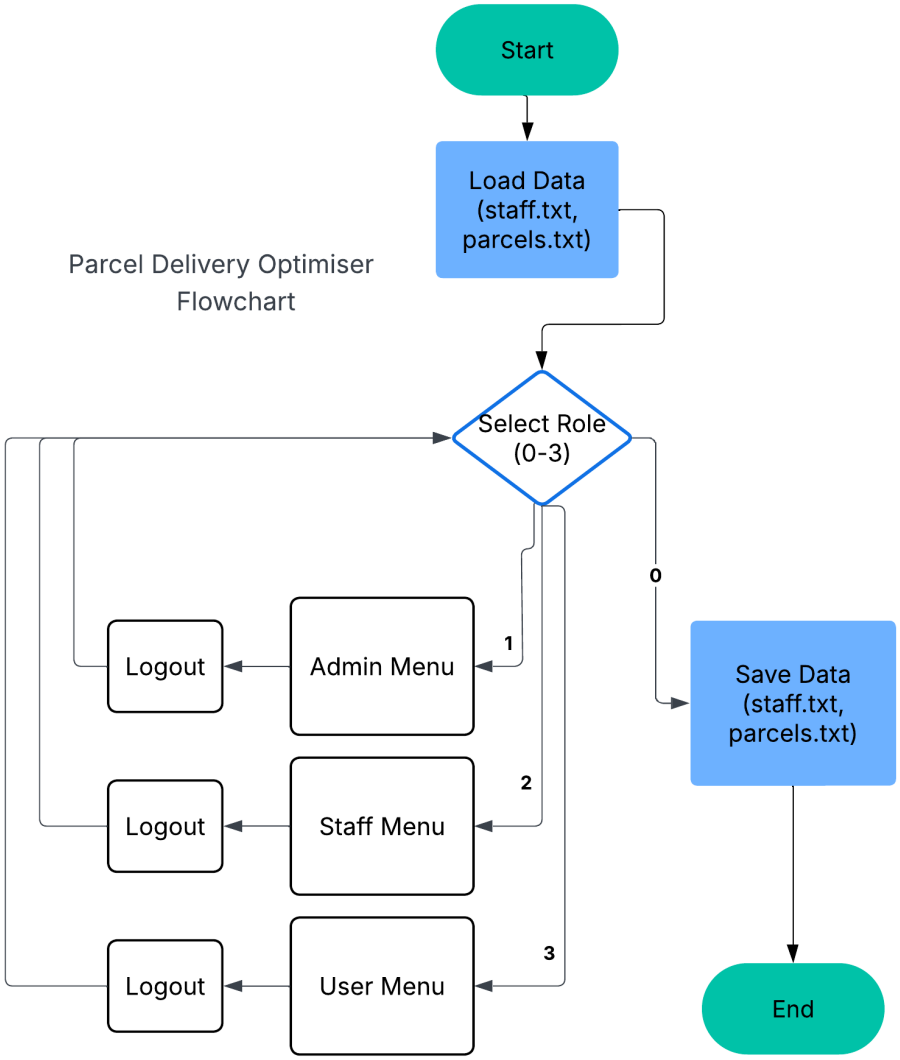


Figure 1: Project Flow Diagram

7 Implementation Details

The Parcel Delivery Optimizer is a C++ console application built with modular functions for algorithms, file handling, and user interface, using robust data structures.

- **User and Staff Menus**: The User menu enables parcel tracking and cancellation, while the Staff menu allows staff to view parcels assigned to their ID. These menus provide essential role-based functionality but are not elaborated in this report, as the **Admin menu** includes the main system operations(algorithms).
- **Algorithms**:
  - **Merge Sort**: Sorts parcels by `distance` for route planning. The `mergeSort` function recursively divides the parcel vector, while `merge` combines sorted subarrays by comparing `distance`.

```
void mergeSort(vector<Parcel>& arr, int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
```

  - **Greedy Scheduling**: Sorts parcels by earliest `deliveryTime` using Earliest Deadline First (EDF) in `greedyAssignParcels`, then assigns them to staff greedily based on this order with Round-Robin for fair distribution across staff.

```
// Sort parcels by earliest delivery time (EDF)
sort(unassignedParcels.begin(), unassignedParcels.end(),
     [](Parcel* a, Parcel* b) { return a->deliveryTime < b->
        deliveryTime; });
```

```
// Assign parcels greedily with Round-Robin
for (Parcel* parcel : unassignedParcels) {
    parcel->assigned = true;
    parcel->assignedStaffId = staffList[staffIndex].id;
    staffIndex = (staffIndex + 1) % staffCount; // Round-
       Robin
}
```

  - **Knapsack (Dynamic Programming)**: The `dpKnapsack` function maximizes total parcel weight within a 10kg capacity using a 2D dynamic programming table. It builds a table to track the maximum weight achievable for each parcel and capacity, choosing to include or exclude parcels based on weight constraints, and displays the result.

```
for (int i = 1; i <= n; i++)
    for (int w = 0; w <= capacity; w++)
        if (parcels[i-1].weight <= w)
            dp[i][w] = max(parcels[i-1].weight + dp[i-1][w -
                parcels[i-1].weight], dp[i-1][w]);
```

- **File Handling**: Functions `saveStaffData` and `saveParcelData` store data in `staff.txt` and `parcels.txt` using a fixed-width tabular format with separators (e.g., `+----...+`). Functions `loadStaffData` and `loadParcelData` parse these files with error handling.
- **User Interface**: Role-specific menus (`adminMenu`, `staffMenu`, `userMenu`) use `centerText` and `printLine` for a clear console layout.
- **Other Implementation**: The system uses `vector<Parcel>` and `vector<Staff>` for dynamic storage. Input handling in `addNewStaff` and `addNewParcel` uses `cin.ignore()` to manage input buffers, but lacks robust validation for non-numeric inputs, a limitation addressed in future scope.

The full source code is available in the GitHub repository (see Appendix).

8 Output Analysis

The Parcel Delivery Optimizer generates console and file outputs to facilitate user interaction and data reporting, designed for clarity and accessibility by users and external stakeholders.

- **Console Output**: The system displays interactive menus and operation results with centered text and dashed line separators for a user-friendly interface. For example, the main menu lists role options (Admin, Staff, User), and the Admin's "View Parcels" function shows parcel details such as ID, status, receiver, distance, and weight. A sample console output is:

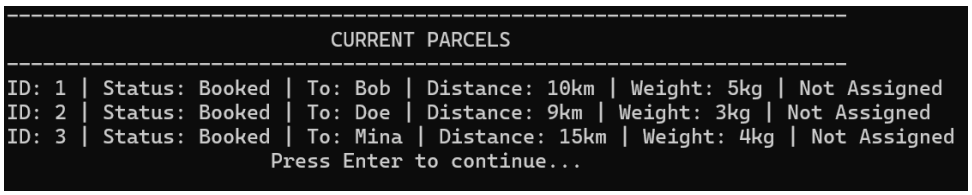


Figure 2: Console Output Showing Initial Parcel Data

The Greedy Assignment result shows parcels assigned to staff by earliest delivery time, here's a snippet:

Figure 3: Console Output – Greedy Assignment by Round-Robin and View Parcels

These outputs ensure intuitive navigation for users (see screenshots at `https://github.com/afrahackerman/Parcel-Delivery-Optimiser` for more).

A second Greedy Assignment uses delivery time (Earliest Deadline First): The Merge

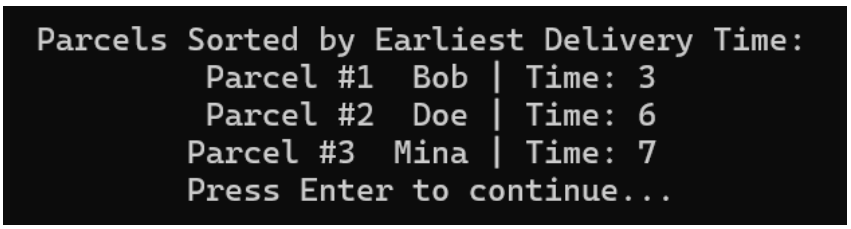


Figure 4: Console Output – Greedy Assignment by Delivery Time (EDF)

Sort output shows parcels sorted by distance:

Figure 5: Console Output – Merge Sort by Distance

The Knapsack output shows optimal parcel selection within weight limits (maximum weights 5+4=9) :

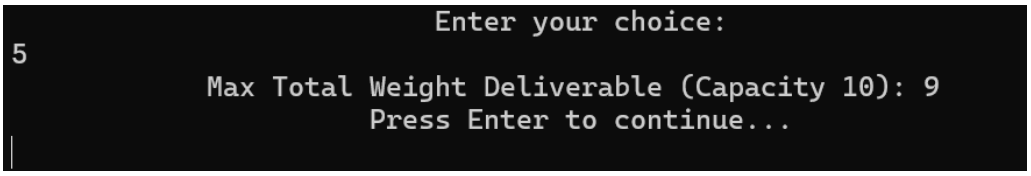
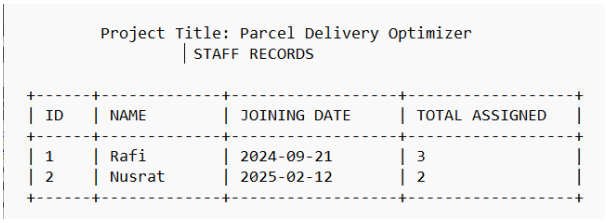


Figure 6: Console Output – Knapsack for Weight-Constrained Delivery

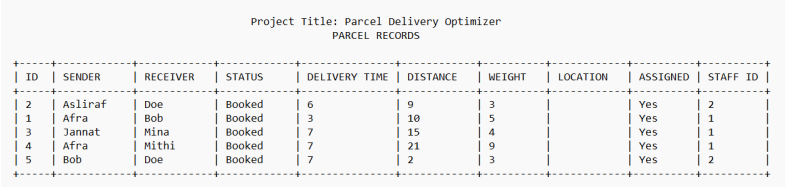
- **File Output**: Data is saved to `staff.txt` and `parcels.txt` in a fixed-width tabular format with headers and separators (`+--...+`). This format ensures readability in text editors like Notepad and compatibility with Excel. Sample outputs of these files are illustrated in Figures 7 and 8.



```
Project Title: Parcel Delivery Optimizer
STAFF RECORDS

+------+-------------+------------------+------------------+
| ID   | NAME        | JOINING DATE     | TOTAL ASSIGNED   |
+------+-------------+------------------+------------------+
| 1    | Rafi        | 2024-09-21       | 3                |
| 2    | Nusrat      | 2025-02-12       | 2                |
+------+-------------+------------------+------------------+
```

Figure 7: `staff.txt` in Notepad



```
Project Title: Parcel Delivery Optimizer
PARCEL RECORDS

+----+----------+----------+--------+---------------+----------+--------+----------+----------+----------+
| ID | SENDER   | RECEIVER | STATUS | DELIVERY TIME | DISTANCE | WEIGHT | LOCATION | ASSIGNED | STAFF ID |
+----+----------+----------+--------+---------------+----------+--------+----------+----------+----------+
| 2  | Asliraf  | Doe      | Booked | 6             | 9        | 3      |          | Yes      | 2        |
| 1  | Afra     | Bob      | Booked | 3             | 10       | 5      |          | Yes      | 1        |
| 3  | Jannat   | Mina     | Booked | 7             | 15       | 4      |          | Yes      | 1        |
| 4  | Afra     | Mithi    | Booked | 7             | 21       | 9      |          | Yes      | 1        |
| 5  | Bob      | Doe      | Booked | 7             | 2        | 3      |          | Yes      | 2        |
+----+----------+----------+--------+---------------+----------+--------+----------+----------+----------+
```

Figure 8: `parcels.txt` in Notepad

- **Analysis**: The console output supports efficient interaction, with centered displays enabling Admins to manage parcels, Staff to update statuses, and Users to track or cancel parcels. The file outputs are optimized for external use, with the fixed-width format ensuring aligned columns in text editors and seamless Excel import. This design makes data accessible to non-technical users like supervisors, meeting the project's goal of transparency.

9 Challenges Faced and Future Scope

9.1 Challenges Faced:

- **Tabular File Format**: Designing a fixed-width format for `staff.txt` and `parcels.txt` required precise column width settings and parsing logic to handle separators (`|`) and ensure alignment in text editors.
- **String Parsing**: Trimming spaces and handling empty fields in `loadStaffData` and `loadParcelData` was complex, resolved using a `trim` function and error checking.
- **Input Validation**: Ensuring robust input handling (e.g., non-numeric IDs) required `cin.fail()` and buffer clearing to prevent crashes.
- **Staff ID Uniqueness**: The system currently allows duplicate staff IDs (e.g., multiple staff with ID 2), which can cause assignment conflicts.
- **Staff Scheduling Conflicts**: The Greedy Assignment algorithm does not prevent a staff member from being assigned multiple parcels with the same delivery time, leading to unrealistic scheduling.
- **Parcel Cancellation Persistence**: When a parcel is canceled during runtime, the system updates its status but does not remove it from the file immediately. This caused inconsistencies until a full save operation was implemented to ensure file synchronization.

9.2 Future Scope:

- **Input Validation**: Add `cin.fail()` checks in `addNewStaff` and `addNewParcel` to validate non-numeric inputs (e.g., staff ID, delivery time), enhancing robustness.
- **Unique Staff IDs**: Implement checks in `addNewStaff` to prevent duplicate staff IDs, ensuring accurate parcel assignment.
- **Staff Availability Check**: Modify `greedyAssignParcels` to ensure no staff is assigned multiple parcels with the same delivery time, preventing scheduling conflicts.
- **Graphical Interface**: Replace console menus with a GUI for easier user interaction.
- **Database Storage**: Use a database (e.g., SQLite) instead of text files for robust data management.
- **Advanced Routing**: Add algorithms like Dijkstra's for optimized delivery paths.

10 Conclusion

The Parcel Delivery Optimizer, a C++ console application, manages logistics with role-based access, optimization algorithms (Merge Sort, Greedy, Knapsack), and readable fixed-width tabular outputs in `staff.txt` and `parcels.txt`. Despite challenges like duplicate staff IDs and scheduling conflicts, it showcases effective C++ programming and is extensible for real-world use.

11 References

References

[1] C++ Standard Library Documentation, `https://en.cppreference.com/w/`.

[2] Cormen, T. H., et al., *Introduction to Algorithms*, 3rd Edition, MIT Press.

12 Appendix

12.1 Source Code Repository

The complete C++ source code for the Parcel Delivery Optimizer is available on GitHub: `https://github.com/afrahackerman/Parcel-Delivery-Optimiser/`

This repository contains:

- Full source code files (`.cpp` and `.h`)
- Sample `staff.txt` and `parcels.txt` data files
- Screenshots of the console in 'pictures' folder
- The flowchart diagram

Setup Instructions:

1. Install Code::Blocks with MinGW compiler.

2. Open `parceldeliveryoptimiser.cpp` in Code::Blocks.
3. Build and run the project.
4. Select a role (Admin, Staff, User) from the main menu.
5. View `staff.txt` and `parcels.txt` in Notepad or Excel for tabular data.

12.2 Sample Console Output

:

```
------------------------------------------------------------------------

                    PARCEL DELIVERY OPTIMIZER
------------------------------------------------------------------------

                   [1] Admin Login
                   [2] Delivery Staff Login
                   [3] User Access
                   [0] Exit
                   Select your role:
```