6CCS3PRJ
Computing Efficient Schedules

Final Project Report

Author: Ubaydullah Naik

Supervisor: Tomasz Radzik

Student ID: 21039338

April 4, 2025

Abstract

This project presents the design and development of a task scheduling platform. It aims to optimise project timelines by minimizing completion time through the use of an advanced scheduling algorithm. The platform retrieves project data from a database, including task details, durations, dependencies, and the number of employees available for the project. Using this information, it estimates the optimal project completion time, generates a corresponding schedule, and provides actionable insights to help users enhance project efficiency. Through this project, organizations can enhance their project management capabilities, improve resource utilization, and ensure timely project delivery.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Ubaydullah Naik

April 4, 2025

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Professor Tomasz Radzik, for his invaluable guidance and support throughout this project. His insightful feedback, encouragement, and expertise have been instrumental in shaping the development of the task scheduling platform. I am deeply appreciative of his time, patience, and dedication, which have greatly contributed to my learning and the successful completion of this project.

Contents

Chapter 1

Introduction

In today's fast-paced and resource-driven world, efficient project management is a critical factor for organisational success [1]. Projects often involve numerous tasks with specific requirements, interdependencies, and resource constraints. Managing these complexities manually can be time-consuming, error-prone, and inefficient, resulting in delays, resource conflicts, and increased costs. The growing demand for intelligent scheduling solutions has become evident across industries such as construction, manufacturing, software development, and event planning [5], where aligning resources with task timelines and ensuring timely completion remains a significant challenge.

Schedulers play a vital role in addressing these issues by automating the scheduling process and providing accurate predictions of project completion time. They enable businesses to remain competitive by ensuring projects are delivered on time and within budget. With optimised plans and actionable insights, schedulers enhance decision-making, facilitate resource management, and minimise the likelihood of project overruns. Additionally, they provide stakeholders with clear timelines and visual representations of task progress, improving transparency and collaboration. By simulating alternative scenarios, schedulers also support contingency planning, empowering project managers to respond effectively to unexpected changes. Ultimately, the implementation of scheduling tools leads to greater accountability, improved operational efficiency, and a significantly higher probability of successful project completion.

1.1 Aims and Objectives

The primary aim of this project is to create an efficient scheduling tool that empowers project managers to make data-driven decisions, avoid bottlenecks, and achieve their goals within the most optimum time frame. The key objectives include:

- Calculating Optimum Completion Time: Using an algorithm to calculate the shortest possible time in which all tasks can be completed whilst considering task requirements.
- Generating Optimised Schedules: Produce actionable schedules that minimise completion time while adhering to resource constraints.
- Providing Visual Representations: Display graphs to visualise the scheduling process.
- Supporting Decision-Making: Offer insights and recommendations to users for further optimising project management.

1.2 Motivation

The motivation for this project arises from the need for a more efficient and reliable scheduling solution that addresses the limitations of manual scheduling and existing tools. Poor scheduling practices can lead to missed deadlines, resource inefficiencies, and financial losses. By implementing a platform that automates and optimises the scheduling process, project managers can gain greater control over task management, reduce the risk of delays, and improve overall project performance. Additionally, the opportunity to apply advanced algorithms to real-world project management challenges makes this project both practically valuable and academically significant.

Chapter 2

Background Research

2.1 Importance Of Project Management

The significance of structured project management in achieving organisational goals and objectives is well-documented. According to the Project Management Institute [3], organisations with robust project management practices are 38% more likely to meet their original goals and 33% less likely to experience project failures. These findings underscore the critical role that effective management plays in aligning project execution with strategic objectives.

A crucial aspect of structured project management is the efficient scheduling of tasks and resources, ensuring that projects progress in a timely and coordinated manner. Schedulers play an integral role in this process by optimising task sequences, minimising delays, and balancing resource constraints. By integrating scheduling tools, organisations can proactively manage dependencies, allocate resources effectively, and mitigate the risks associated with poor planning.

Further supporting this perspective, Aubry et al.'s study highlights the transformative impact of Project Management Offices (PMOs) on organisational efficiency [1]. Their research emphasises the ability of PMOs to balance complex stakeholder dynamics, maintain project performance, and foster a collaborative environment where all contributors feel valued. Schedulers contribute to this structured approach by providing clear timelines that should enable better communication between teams, together, these insights demonstrate how structured project management—bolstered by advanced scheduling techniques—not only ensures successful outcomes but also enhances organisational resilience and adaptability in dynamic environments.

2.2 Existing Tools

Gantt Charts

Gantt charts are one of the most widely used tools in project management for visualising project schedules. A Gantt chart displays tasks as horizontal bars along a timeline, with the length of each bar representing the duration of the task. Tasks are arranged in chronological order, allowing project managers to visualise start and end dates, task sequences, and dependencies. Tasks that cannot begin until others are completed can be linked using dependency lines, providing clarity on project flow. Additionally, Gantt charts often highlight milestones and deadlines, offering a clear overview of critical deliverables. They are particularly effective in tracking task progress, as progress bars can be added to represent completed portions of tasks.

However, Gantt charts have certain limitations. For large-scale projects with numerous tasks and dependencies, they can become overcrowded and difficult to interpret. Maintaining and frequently updating Gantt charts can be labor-intensive, particularly for dynamic projects with frequent changes. Additionally, while Gantt charts are effective for visualising timelines, they do not compute schedules independently but instead rely on pre-existing schedules. Consequently, their effectiveness is directly influenced by the scheduling method implemented for the project. Without a reliable scheduling algorithm or systematic planning, Gantt charts may fail to accurately represent the most optimum way of completing the project.

Kanban Boards

Kanban boards are another widely used project management tool that provides a visual representation of tasks and their progress. They help teams manage workflows by visualising tasks at various stages of completion. A typical Kanban board consists of columns representing different phases of a project, such as ”To Do,” ”In Progress,” and ”Done,” with tasks represented as cards that move from one column to the next as they progress.

One of the main advantages of Kanban boards is their simplicity and adaptability. They offer a clear, real-time view of project status, making them excel in real-time updates and collaboration. By limiting the number of tasks in progress at any given time, Kanban boards prevent overworking team members and promote a steady, manageable workflow. This makes them particularly effective for projects requiring continuous delivery or frequent updates.

However, while Kanban boards are excellent for visualising workflows and improving team collaboration, they may not be as effective for managing projects with complex dependencies or strict timelines. Unlike Gantt charts, Kanban boards do not provide a timeline-based view or an estimate on when the project may be complete. Consequently, they are often used alongside other tools to ensure comprehensive scheduling and progress tracking.

Oracle Primavera P6

Oracle Primavera P6 [2] is a powerful project management solution widely used in large-scale and enterprise-level projects. Developed by Oracle, Primavera P6 is specifically designed for managing complex projects across various industries, including construction, engineering, and manufacturing. It provides advanced scheduling capabilities, resource management, and real-time project monitoring, making it an essential tool for organisations handling extensive and multi-faceted projects.

One of Primavera P6's key strengths lies in its ability to manage large datasets and intricate project networks. Unlike simpler scheduling tools, Primavera P6 offers sophisticated algorithms to calculate optimum schedules, forecast project timelines, and optimise resource allocation. It supports scenario analysis, allowing project managers to simulate different outcomes and make data-driven decisions. Additionally, its robust reporting and analytics features provide detailed insights into project performance, helping stakeholders stay informed and proactively address issues. Primavera P6 also excels in multi-project management, enabling organisations to oversee multiple projects simultaneously. Its resource allocation features ensure that resources are distributed efficiently across projects, reducing the risk of overallocation or underutilisation.

However, the complexity and cost of Primavera P6 can be a drawback for smaller businesses or projects with limited budgets. The platform requires significant training and expertise to use effectively, and its implementation can be time-consuming. Additionally, its extensive features may be excessive for simpler projects that do not require detailed scheduling and resource management.

2.3 Algorithms

Shortest Processing Time

The Shortest Processing Time (SPT) algorithm is a simple and efficient scheduling technique that prioritises tasks with the shortest processing time first. This algorithm is often applied in production environments and scenarios where reducing average completion time is a priority. By processing shorter tasks first, SPT minimises overall waiting time and reduces the average completion time across all tasks. One of its primary advantages is its computational simplicity, making it easy to implement and effective for smaller projects with independent tasks.

However, the SPT algorithm has notable limitations. It may lead to issues of fairness, as tasks with longer processing times are consistently deprioritised, causing significant delays for those tasks. This phenomenon, known as ”starvation,” is especially problematic in environments where long tasks are critical to project completion. Additionally, SPT does not consider task dependencies or resource constraints, making it unsuitable for complex projects where task relationships must be accounted for. While SPT is beneficial for optimising completion times in straightforward scenarios, its limitations make it less effective in managing large-scale or resource-intensive projects.

Longest Path

One of the primary advantages of using the longest path algorithm in a Directed Acyclic Graph (DAG) is its ability to accurately compute project timelines by identifying tasks most likely to delay the project. The longest path in a DAG is known as the critical path, where any delay in the tasks along this path will directly impact the overall project completion time. In contrast, tasks not on the critical path typically have some degree of flexibility, allowing project managers to focus resources and attention on the critical tasks to ensure timely completion.

However, the longest path algorithm has certain limitations. It assumes that task durations are known and fixed, which may not be realistic in dynamic or uncertain project environments. Additionally, the algorithm operates under the assumption of unlimited parallel processing power, meaning it presumes all tasks without dependencies can be executed simultaneously. In practice, projects are constrained by finite resources, such as labour, equipment, or materials, making this assumption impractical. Furthermore, while the longest path algorithm identifies the overall project duration, it does not provide a detailed, time-based schedule, which can make it difficult for managers to visualise task sequences and allocate resources effectively.

Simulated Annealing

The Simulated Annealing algorithm is a probabilistic optimisation technique inspired by the annealing process in metallurgy, where materials are heated and gradually cooled to reduce defects and improve structural integrity. In the context of scheduling, Simulated Annealing is used to find near-optimal solutions by exploring a wide range of possible schedules and progressively refining the results.

A key advantage of Simulated Annealing is its ability to find the global optimum by accepting worse solutions with a certain probability, particularly in the early stages of the algorithm. This characteristic makes it highly effective for solving complex scheduling problems where traditional algorithms may become trapped in suboptimal solutions. Additionally, Simulated Annealing is flexible and can be applied to a variety of scheduling scenarios, including those with resource constraints, task dependencies, and dynamic changes. Its simplicity and adaptability also make it suitable for large-scale problems where exhaustive search methods would be computationally impractical.

However, Simulated Annealing also has limitations. The algorithm's performance is highly dependent on the choice of parameters, including the cooling schedule and the acceptance probability. Poor parameter selection can lead to slow convergence or failure to find a sufficiently optimal solution. Furthermore, since it is a stochastic algorithm, results may vary across different runs, and there is no guarantee of reaching the absolute optimal solution. The computational cost can also be high for larger problems, especially when numerous iterations are required to explore the solution space effectively.

Reinforcement learning

Reinforcement Learning (RL) is an artificial intelligence technique that uses trial-and-error learning to determine optimal decision-making policies. In the context of scheduling, RL algorithms interact with the environment by making scheduling decisions, receiving feedback in the form of rewards or penalties, and using this information to refine future decisions. This makes RL particularly effective for dynamic and complex scheduling problems where uncertainties exist, and schedules may need to be frequently adjusted.

One of the primary advantages of RL is its adaptability. Unlike traditional scheduling algorithms that rely on static data and fixed assumptions, RL continuously learns and improves its decision-making process based on real-time data. This capability is especially beneficial in industries such as manufacturing, logistics, and cloud computing, where unforeseen disruptions or changes in resource availability are common. Additionally, RL algorithms can optimise multiple objectives simultaneously, such as minimising project completion time, reducing costs, or maximising resource utilisation.

However, RL also has limitations. The learning process can be computationally expensive and time-consuming, particularly for large-scale scheduling problems with numerous tasks and constraints. The algorithm requires extensive data to train effectively, and poor reward design can lead to suboptimal scheduling policies. Additionally, since RL decisions are based on historical interactions, the algorithm may struggle in entirely new scenarios without sufficient training data. Furthermore, the interpretability of RL models can be a challenge, as the reasoning behind specific decisions may not always be transparent.

2.4 Findings

The findings from this research reveal significant gaps in the effectiveness of existing scheduling tools and algorithms in real-world applications. A major limitation is their reliance on static inputs and assumptions, with many algorithms operating under the expectation that task durations are fixed and predictable. In practice, this assumption is often unrealistic, particularly in dynamic environments where unforeseen changes can disrupt project timelines. Additionally, while smaller projects may benefit from simpler algorithms and visual tools, traditional methods struggle to handle the complexity of large-scale projects involving hundreds of interdependent tasks. Algorithms may become computationally inefficient, leading to delays in generating schedules, while visual tools like Gantt charts can become cluttered and difficult to interpret.

Furthermore, solutions that offer scalability tend to be prohibitively expensive and often come with a steep learning curve, limiting their accessibility for smaller organisations. On the other hand, AI-based algorithms such as RL introduce adaptability and improved decision-making but often lack transparency. The inability to provide clear insights into how scheduling decisions are made undermines trust and complicates the justification of choices to stakeholders.

This project seeks to address these limitations by developing a platform capable of managing uncertainty, scaling effectively with complex task dependencies, and optimising resource management. Additionally, it aims to enhance transparency by offering actionable insights into project timelines and identifying areas for improvement. By providing a more adaptive and interpretable scheduling solution, the platform can assist project managers in making more informed decisions, reducing project risks, and improving overall efficiency.

Chapter 3

Design & Specifications

3.1 Algorithm

Justification

The Longest Path Algorithm is the most suitable choice for this scheduling platform as it explicitly accounts for task dependencies and provides a transparent critical path for project managers. Unlike other approaches, such as the Shortest Processing Time (SPT), Reinforcement Learning (RL), and Simulated Annealing (SA), the Longest Path Algorithm ensures that the overall project timeline is accurately determined and that critical tasks are clearly identified.

SPT, while efficient for minimising individual task waiting times, does not consider dependencies, making it unsuitable for complex scheduling. RL, though adaptive, requires extensive computational resources and lacks transparency, which limits its practical use in project management. SA, being a probabilistic optimisation method, does not inherently prioritise critical paths and may not always produce the most reliable schedule.

However, the traditional Longest Path Algorithm has three main drawbacks: it assumes fixed task durations, lacks resource management, and does not provide a detailed schedule. This project aims to address these limitations by incorporating mechanisms to handle task duration uncertainty, integrating resource constraints to ensure realistic scheduling, and enhancing output clarity with a more structured and detailed schedule. By improving upon these aspects, the platform will offer a more practical and effective scheduling solution while retaining the benefits of critical path analysis.

3.2 System Overview

The scheduling platform is designed as a unified backend system that processes project data stored within a database to generate optimised project schedules. It models tasks and their dependencies using a Directed Acyclic Graph (DAG), where each node represents a task and each edge signifies a dependency between tasks. This graph structure enables the system to efficiently compute task sequences, identify critical paths, and optimise the overall project timeline.

At the heart of the platform lies the longest path algorithm, which has been adapted to handle task duration uncertainty and resource constraints. The algorithm calculates the optimal sequence of tasks and identifies the critical path. In addition to the critical path, the platform is capable of addressing other task relationships, such as mutually exclusive tasks, by incorporating those adjustments into the DAG structure.

Once the scheduling calculations are complete, the platform generates output in the form of a visualisation of the DAG, along with a daily schedule for resource allocation, particularly for employees. These outputs provide valuable insights into the project, highlighting areas that require attention and offering a clear roadmap for task execution. This combination of data-driven scheduling and visual representation ensures project managers have a comprehensive view of the project's timeline, dependencies, and resource allocation.

3.3 Software Framework

Django is an ideal framework for building a scheduling platform due to its rapid development capabilities, scalability, and strong security features. It includes built-in tools, such as authentication, an admin interface, and database management, to allow for quick implementation of core functionalities like task management and resource allocation. Django's Object-Relational Mapping (ORM) simplifies managing complex relationships between tasks, resources, and dependencies.

Django's scalability is another key advantage, as it can efficiently handle large projects and numerous users. Additionally, Django has strong security features, including protection against common vulnerabilities and a robust user authentication system, ensuring the platform's safety. The framework's large community and extensive ecosystem of third-party libraries make it a reliable choice for long-term maintenance and feature expansion.

3.4 Python Packages

NetworkX

NetworkX is a widely used Python library designed for creating, manipulating, and analysing complex graph structures. It provides an intuitive and efficient framework for handling directed graphs, allowing for the representation of tasks as nodes and dependencies as edges. NetworkX includes built-in algorithms for computing detecting cycles and analysing graph properties, which are essential for accurately modelling project schedules. Its ability to handle large-scale graphs efficiently ensures that the scheduling algorithm remains computationally feasible even for complex projects.

Matplotlib

Matplotlib is a popular data visualisation library that complements NetworkX by allowing users to generate clear and informative graphical representations of the DAG. By visualising the task dependencies in a structured format, project managers can gain valuable insights into the scheduling constraints and critical paths. The ability to render DAGs with labelled nodes and directed edges makes it easier to interpret the relationships between tasks, aiding in decision-making and schedule optimisation.

3.5 Data Management

Effective data management is crucial for ensuring accurate scheduling and resource allocation. The platform relies on structured project data stored in a database, which serves as the foundation for task scheduling and dependency management. The key data components include:

- Project Information: Each project is assigned a name and description to facilitate organisation and identification. Additionally, the project start date is recorded to establish the scheduling timeline.
- Task Data: Each task within a project is assigned a unique Task Name or ID to ensure accurate tracking. The duration of each task is recorded in days, which can either be a fixed value or an estimated range (e.g., 3–5 days) to account for uncertainties in project execution.
- Dependency Relationships: Tasks often have dependencies that determine their execution order. For example, if Task B depends on Task A, Task B cannot begin until Task A

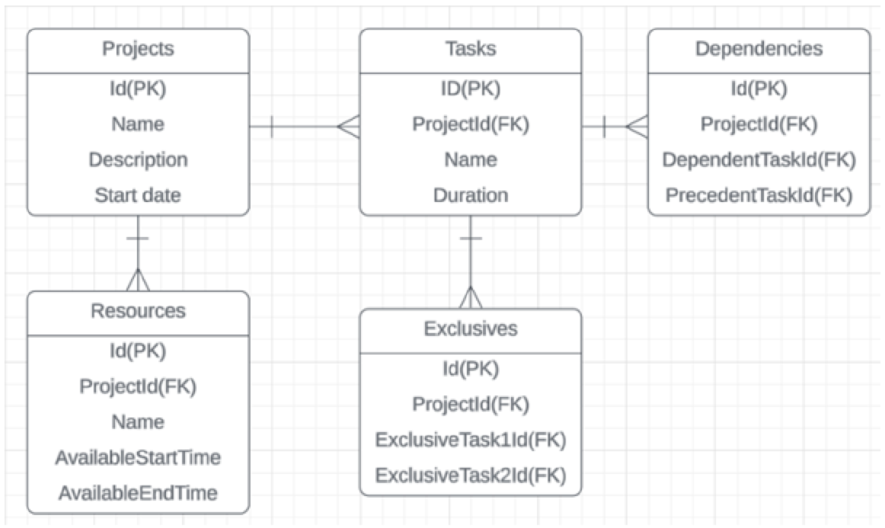


Figure 3.1: An Entity Relationship Diagram

is completed. These relationships are essential for constructing the project's Directed Acyclic Graph (DAG) and computing the critical path.

- Mutually Exclusive Relationships: Some tasks may not be performed simultaneously due to constraints such as shared resources, workspace limitations, or task conflicts. The system records pairs of mutually exclusive tasks to ensure that they are scheduled accordingly.
- Available Resources: The number of employees assigned to the project is a critical factor in scheduling. Each employee can only work on a single task per day, meaning resource availability directly impacts task execution and project duration. The platform considers these constraints when generating the schedule to ensure efficient workforce allocation.

The way the data will be managed is illustrated in the Entity-Relationship diagram in Figure 3.1

Considerations And Constraints

To ensure the accuracy and reliability of the scheduling platform, several key constraints and considerations have been implemented within the database design.

One of the most critical aspects of data integrity is referential integrity, which ensures that relationships between tables remain valid and consistent. Foreign keys are used to enforce these relationships, preventing the creation of orphaned records. For instance, each task must be associated with a valid project, and all dependencies and exclusivity constraints must reference existing tasks within the same project. Additionally, when a project is deleted, all associated tasks, dependencies, and exclusivity constraints must be handled appropriately. This could be achieved through cascading deletions, where all related records are automatically removed, or through restricted deletion policies that prevent deletion if dependent records exist.

Another crucial constraint is the validation of task dependencies. The system must prevent circular dependencies, which could result in infinite scheduling loops. A task should not depend on another task that ultimately relies on the original task. For example, if Task A depends on Task B, Task B depends on Task C, and Task C depends on Task A, this would create an invalid cycle. The scheduling algorithm should include checks to ensure that such cycles do not exist before processing the schedule.

Scalability is also a significant consideration, particularly in managing mutually exclusive tasks. The current database structure assumes that exclusivity constraints apply between pairs of tasks. However, if more than two tasks must be mutually exclusive, the existing model may require modification. One possible solution would be to introduce an ExclusiveGroups table, where a group of tasks can be marked as mutually exclusive, rather than defining exclusivity at the pair level. This would improve flexibility and allow for more complex exclusivity constraints to be handled efficiently.

3.6 Requirements

The requirements for the scheduling platform are divided into three main categories: user requirements, functional requirements, and non-functional requirements. These collectively define what the system must achieve, how it should operate, and the constraints within which it must function.

User Requirements

User requirements define what the end users expect from the system. Since the platform does not have a user interface, its primary users are assumed to interact with it indirectly through the database or generated outputs. The key user requirements include:

- The system must generate optimised project schedules based on task dependencies and resource constraints.
- The system must provide a clear breakdown of the project timeline, highlighting the critical path.

- The system must output meaningful insights regarding project structure and scheduling improvements.

Functional Requirements

Functional requirements specify the system's behaviour and define what operations it must perform. The key functional requirements include:

- The system must retrieve project data from a database, including tasks, dependencies, mutually exclusive tasks, and available resources.
- The system must construct a Directed Acyclic Graph (DAG) to represent the task dependencies.
- The system must check for and prevent circular dependencies before processing the schedule.
- The system must use a modified longest path algorithm to determine the critical path and estimate the minimum project completion time.
- The system must generate a task execution plan, outlining which tasks should be performed on each day.
- The system must ensure that mutually exclusive tasks are never scheduled on the same day.
- The system must consider resource availability when scheduling tasks, ensuring that no more tasks are assigned than there are employees available.
- The system must output:
  - A visualisation of the DAG, clearly indicating the critical path.
  - A structured daily schedule for employees.
  - Insights into possible scheduling improvements.

Non-Functional Requirements

Non-functional requirements define the quality attributes and operational constraints of the system. The key non-functional requirements include:

- Performance: The system should generate schedules efficiently, even for large-scale projects with many tasks and dependencies.
- Scalability: The system should be capable of handling projects of varying complexity without significant performance degradation.
- Reliability: The system must ensure that no invalid schedules (e.g., those violating dependencies or resource constraints) are produced.
- Maintainability: The system should be designed in a way that allows easy modifications to the scheduling algorithm or database structure if needed.
- Accuracy: The algorithm should function as a heuristic, aiming for near-optimal schedules while not guaranteeing the absolute best solution. It may find the optimal schedule in some cases but prioritises feasibility and efficiency.
- Data Consistency: The system must ensure that any updates to project data are reflected correctly in the generated schedule.

Chapter 4

Implementation

4.1 Django Models

The project was initialised using Django, a high-level Python web framework that provides an ORM (Object-Relational Mapping) system for managing database interactions. Models serve as a foundation for this system, and they allow developers to define database structures using Python classes rather than writing raw SQL. Each model in Django is a Python class that maps to a database table, where attributes represent fields and their properties. One of the key advantages of Django models is automatic database schema generation and management. By defining models in Python, Django can automatically create and modify database tables using migrations, which track changes to the schema over time. This ensures that database structures evolve with the project without requiring manual intervention.

In this project, Django models were used to represent projects, tasks, employees, dependencies and exclusions. The ORM allowed for efficient querying and manipulation of data, ensuring that scheduling computations could be performed dynamically without writing complex SQL queries.

4.2 Seeding The Database

Since the scheduling platform is designed to extract data from a database, the database must be populated with test data to validate its functionality. The test data is based on Figure 4.1, where nodes such as *begin, a, b, c, etc.*, represent individual tasks. Each task has an associated duration, indicated by the numbers above the nodes, while the directed edges between nodes

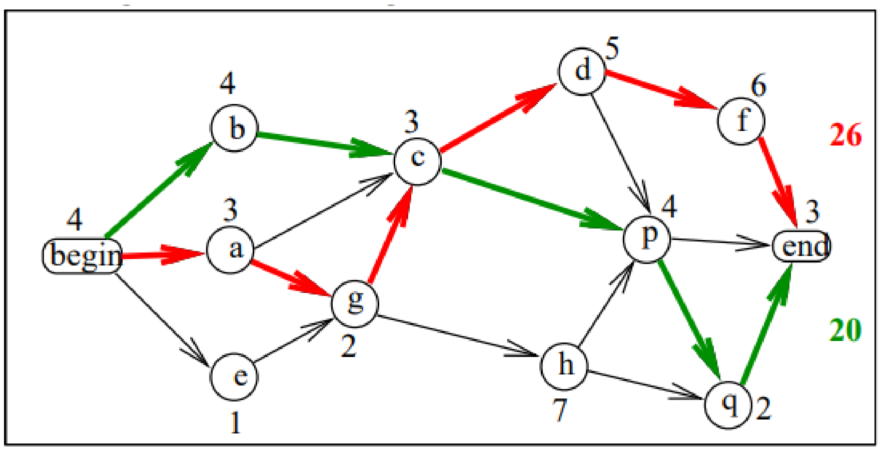


Figure 4.1: An example of a DAG used for the Longest Path Problem [4]

represent task dependencies. This example also highlights the critical path and its length in red, which provides a clear benchmark for the expected output.

Seeding was performed using Django's seed command, where instances of the Project, Task, and Dependency models were created and stored in the database. Since the test data was based on Figure 4.1, the Employee and Exclusion models were not required. Utilising the seed command instead of manually entering data into the database allows for quick and consistent regeneration of test data, facilitating iterative development and ensuring reproducibility across different testing environments.

4.3 Generating A DAG

Before applying the longest path algorithm, a Directed Acyclic Graph (DAG) must be constructed using the data stored in the database. The NetworkX package in Python was chosen for this task due to its built-in functions that simplify graph creation and manipulation.

The process begins by retrieving all dependency relationships for the given project from the database. Using NetworkX functions, directed edges are added between dependent tasks. If a task has not been previously added, the function automatically generates the corresponding node. Next, tasks that have no dependencies are identified by filtering the database accordingly. Since a DAG requires a single entry point, a Source node is introduced to represent the starting point of the project. This node is then connected to all tasks that have no dependencies. The weight of each edge is assigned based on the duration of the dependent task, ensuring that the graph accurately represents task durations and dependencies. Finally, to maintain the integrity of the scheduling process, the constructed DAG is checked for cycles. Any cycles present would indicate a circular dependency, making scheduling impossible.

4.4 Implementing The Longest Path Algorithm

To determine the longest path in the DAG, a topological ordering of the nodes is first generated. This ensures that each task is processed only after all its dependencies have been considered.

Two dictionaries are then initialised. The first dictionary stores each node's distance from the source node, representing the longest time required to reach that task. The second dictionary records each node's predecessor, which is the previous task in the longest path leading to that node. The predecessor mapping allows for backtracking to reconstruct the critical path.

All nodes are initially assigned a distance of -1, except for the source node, which is set to 0 to indicate the starting point of the project. The predecessors of all nodes are also set to -1, as no paths have been determined yet.

The algorithm then iterates through all nodes in topological order. For each node u, all adjacent nodes v are retrieved. If the currently stored distance of v is less than the distance of u plus the edge weight from u to v, the distance of v is updated to reflect the new longest known distance, and u is recorded as the predecessor of v. This process ensures that each node's distance reflects the longest time required to reach it.

Once all nodes have been processed, the node with the largest distance in the dictionary represents the shortest possible completion time, as well as the task at the end of the longest path. By tracing backwards through the predecessor dictionary from this node to the source node, the full sequence of tasks in the longest path can be identified. Reversing this sequence yields the correct order of tasks along the longest path, representing the critical path of the project.

4.5 Visualising The Graph

To generate a visualisation of the project's Directed Acyclic Graph (DAG) and highlight the longest path, the NetworkX and Matplotlib libraries were used. The process begins by giving the graph a logical structure. Each node is assigned a specific layer based on its position in the topological order. Nodes that have no dependencies are placed at the first level, while subsequent tasks are positioned in layers corresponding to their dependency depth. This ensures that the visualisation maintains a clear flow from the project's start to completion.

Once the node positions are determined, the graph is drawn with all tasks labelled with

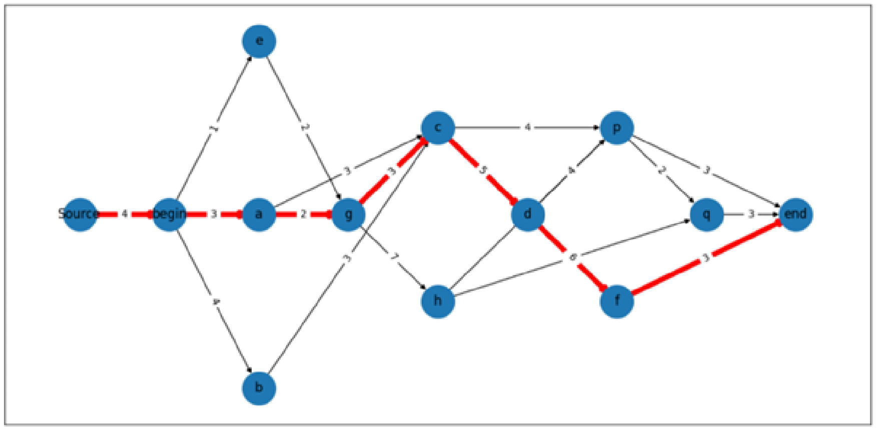


Figure 4.2: A visualisation of the DAG as program output

their names, with each edge labelled with its corresponding task duration to provide clarity on the project timeline. To emphasise the longest path, the edges forming this path are identified and redrawn with increased width and distinct colour, making them stand out from the rest of the dependencies. Finally, the visualisation is refined by adjusting the layout for readability, ensuring that nodes do not overlap and that all labels remain visible. The output of the process is shown in Figure 4.2, where the critical path is shown correctly, similar to how it was shown in the example in Figure 4.1. This indicates that both the algorithm and the visualisation process function as intended.

4.6 Handling Mutual Exclusions

Solution

In some cases, two tasks may not be directly dependent on each other but still cannot be executed simultaneously due to resource constraints or other limitations. The standard longest path algorithm assumes that all tasks without dependencies can begin at the same time, which means it does not account for mutually exclusive tasks. As a result, the algorithm may incorrectly schedule such tasks concurrently, leading to an unrealistic project timeline. To address this, an artificial dependency is introduced between mutually exclusive tasks. This ensures that one task is always scheduled before the other, preventing them from running in parallel.

Determining Task Order

When enforcing mutual exclusivity between two tasks, a decision must be made on which task should precede the other. One approach is to generate two versions of the project graph, each with a different ordering of mutually exclusive tasks, and compute the longest path for both. The configuration that results in the shortest project completion time is then selected as the more optimal schedule.

While this method works well when there is only one pair of mutually exclusive tasks, it becomes less effective when multiple such constraints exist. The approach evaluates each mutual exclusion constraint in isolation rather than considering all constraints collectively. For example, if Task A and Task B are mutually exclusive, and Task C and Task D are also mutually exclusive, the method may find a locally optimal ordering (e.g., A $\rightarrow$ B) but fail to identify a globally optimal configuration where B $\rightarrow$ A and C $\rightarrow$ D together produce a shorter project schedule.

Attempting to find the absolute global optimum would require evaluating all possible orderings of mutually exclusive tasks, which grows exponentially as $2n$ for n mutually exclusive pairs. This becomes computationally infeasible for large projects. Instead, the system relies on the heuristic of selecting the local optimum at each step, which provides a reasonable balance between efficiency and accuracy while ensuring that mutually exclusive constraints are respected in the scheduling process.

4.7 Managing Resource Constraints

The Problem

The longest path algorithm assumes that any number of tasks can be executed simultaneously, provided they have no prior dependencies. However, real-world projects operate under resource constraints, particularly with a limited number of employees available to work on tasks. This assumption can lead to unrealistic scheduling outcomes.

For example, consider the project structure in Figure 4.3, where all edge weights are 1. The algorithm would schedule Tasks A, B, C, and D simultaneously in the first time unit, followed by Tasks E and F in the second unit, Tasks G, H, and I in the third unit, and Task J in the fourth unit. This results in the shortest possible completion time of four units. However, this schedule assumes that all four initial tasks can be executed at once. If the project has only three employees, this would be infeasible, and the schedule must be adjusted accordingly.

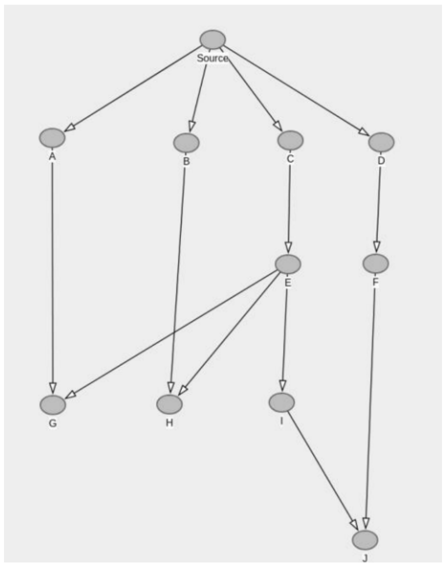


Figure 4.3: A DAG used as an example for resource management

Adjusting The Schedule

Since each layer of the graph represents tasks that can be executed concurrently, modifying the topological ordering of the DAG allows for better resource management. By introducing artificial dependencies, the number of concurrently executed tasks is limited to match the available workforce. Tasks that exceed the employee limit are deferred until workers become available, ensuring that no more tasks are scheduled simultaneously than the project can accommodate.

Using Figure 4.3 as a reference and assuming only three employees are available, the adjusted DAG in Figure 4.4 demonstrates a possible solution. In this revised graph, each topological layer contains no more than three tasks at a time, aligning with the available workforce. This adjustment produces a more practical schedule, enabling project managers to allocate resources effectively while maintaining an efficient project timeline.

Determining Task order

The choice of which tasks to defer is crucial in minimising the overall project completion time. As seen in Figure 4.4, linking Task B to Task A does not extend the longest path. However, if Task B were linked to Task C instead, the longest path would increase from 4 to 5. To optimise scheduling while enforcing resource constraints, tasks in a given layer are examined to determine if their count exceeds the number of employees. If so, each task is considered as a potential starting point, and a subgraph is constructed for it. The longest path for each subgraph is then computed, and an artificial dependency is introduced from the task with the shortest path to

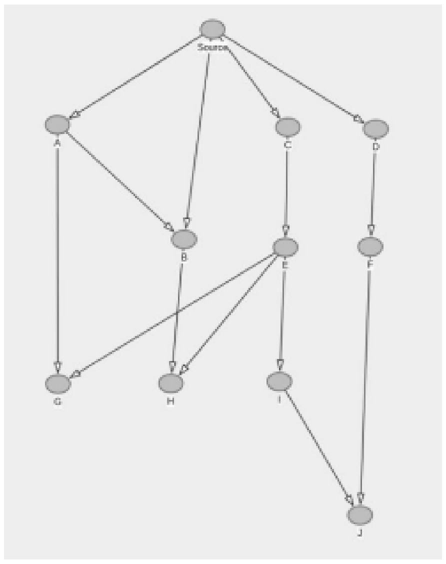


Figure 4.4: A possible solution to the resource problem

the task with the second-longest path. This balances the workload while minimising the overall project duration.

This approach provides a heuristic solution rather than an exact optimal one. Ideally, the most efficient schedule would be found by evaluating all possible ways to reorder tasks while still satisfying dependency constraints. However, this would require testing every possible combination, leading to a factorial growth in computational complexity. Since this is infeasible for large projects, the method used provides a reasonable estimate of the optimal schedule while ensuring practical runtime efficiency.

4.8 Uniform Graphs

The Issue With Non Uniformity

A significant limitation of the current resource management approach is that it assumes each topological layer represents a single unit of time. This assumption holds only when all tasks have the same duration, which is an unlikely scenario in real-world projects. When task durations vary, this approach can produce inaccurate scheduling results.

For example, in Figure 4.5, assuming that two employees are available, the shortest completion time is calculated as 4. However, this estimate does not accurately reflect the actual time required to complete the project. While one employee spends 4 days working on Task B, the other is expected to complete Tasks A, C, D, and E. The combined duration of these tasks

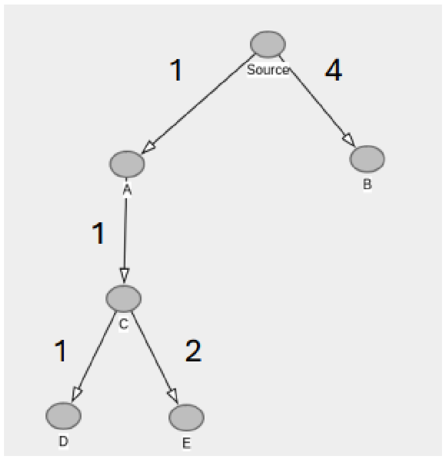


Figure 4.5: An example of a Non-Uniform DAG

is 5 days, meaning that the scheduling method does not fully account for employee workload distribution. Even though each layer adheres to the employee constraint by not exceeding the number of available employees, the algorithm does not consider the varying durations of tasks, leading to an unrealistic project timeline.

Creating A Uniform DAG

The solution to this issue is to transform the non-uniform graph into a uniform one, ensuring that each layer accurately represents concurrent tasks. This is achieved by breaking down tasks with durations greater than one into multiple smaller tasks, each representing a single unit of time. For a task with a duration of n, where n ¿ 1, it is replaced by n sequential sub-tasks.

To maintain dependencies, all incoming edges to the original task are redirected to the first of the newly created sub-tasks, while all outgoing edges originate from the last sub-task in the sequence. This transformation ensures that all edges in the graph represent uniform time intervals while preserving the original dependencies. Figure 4.6 illustrates this approach, applying it to the graph from Figure 4.5 as input.

With the graph now transformed into a uniform representation where each layer corresponds to a single unit of time, the resource management method can be applied effectively. By enforcing resource constraints on this modified graph, the resulting schedule accurately reflects real-world limitations. Figure 4.7 presents the adjusted version of the graph.

In this new graph, the longest path is calculated to be 5, which correctly represents the minimum time required to complete the project with only two employees. Unlike the original non-uniform graph, this approach ensures that resource constraints are properly enforced,

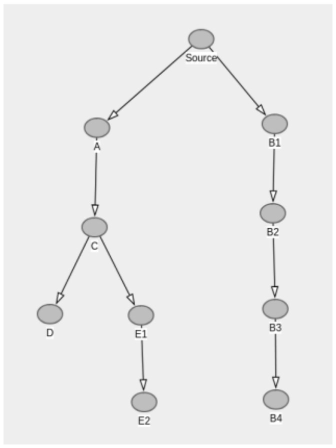


Figure 4.6: An example of a Uniform DAG

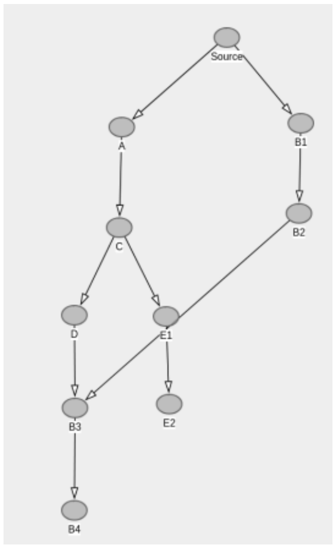


Figure 4.7: The Solved Resource Management DAG

preventing unrealistic task allocations and leading to a more feasible project schedule.

Creating The Schedule

An additional advantage of transforming the graph into a uniform representation is that it naturally provides a structured daily schedule for the project. Since each layer in the graph corresponds to a single unit of time, iterating through the graph layer by layer allows for the extraction of tasks assigned to each day. This ensures that project managers can clearly see which tasks need to be completed on each specific day to achieve the shortest possible completion time. By following this structured schedule, teams can efficiently allocate resources and track progress while ensuring that all dependencies and constraints are met.

4.9 Providing Insights

The platform provides valuable insights that help project managers assess the impact of resource constraints on project completion time and make informed decisions about resource allocation.

One key insight is the comparison between the optimum completion time, assuming unlimited resources, and the completion time under the project's actual resource constraints. By calculating both values, project managers can quantify the severity of resource bottlenecks and determine how much efficiency is lost due to limited staffing. This allows them to evaluate whether increasing the number of employees would yield significant improvements in project completion time.

Another important insight is determining the minimum number of employees required to achieve the optimal completion time. This is done by first converting the project into a uniform graph, where the maximum number of concurrent tasks in any layer is identified as a potential upper bound on the required employees. The resource management process is then applied iteratively, reducing the number of employees one at a time. If reducing the number of employees does not increase the longest path, the process continues until the longest path does increase. The smallest number of employees that maintains the optimal completion time is identified as the minimum resource requirement. This helps project managers strike a balance between minimising project duration and controlling resource costs.

4.10 Handling Uncertainty

One of the key limitations of the standard longest path method is its reliance on fixed task durations. In real-world projects, task durations are often uncertain and can only be estimated within a range. To address this, the platform incorporates uncertainty by allowing for variable task durations and analysing both best-case and worst-case scenarios.

To achieve this, modifications are made to the database schema by replacing the single task duration value with two fields: *min_estimate* and *max_estimate*, representing the lower and upper bounds of the estimated task duration. Using this data, two separate graphs are constructed—one where all tasks take their minimum estimated duration and another where all tasks take their maximum estimated duration.

Each of these graphs undergoes the same processing steps: managing mutual exclusions, converting into a uniform graph, handling resource constraints, generating a schedule, and providing insights. Due to the differences in edge weights between the two graphs, the results may vary significantly. The worst-case scenario graph may not only have a longer completion time but may also require additional employees to achieve an optimal schedule.

By providing two schedules—one based on the best-case estimates and another based on the worst-case estimates—project managers gain a clearer understanding of the possible range for project completion (e.g., 50–70 days). This enables better decision-making regarding risk mitigation. For time-sensitive projects, managers can take proactive steps to minimise the worst-case estimates, such as investing in better tools or assigning more experienced employees. Additionally, when setting delivery expectations with clients, using the worst-case completion time helps prevent overpromising and ensures more realistic commitments.

Chapter 5

Evaluation & Testing

5.1 Testing Methodology

Testing was conducted manually by running the scheduling platform with different project datasets and verifying whether the generated schedules aligned with expected outputs. The test cases were designed to assess the correctness of the scheduling logic, handling of constraints (dependencies, mutual exclusions, resource limits), and the ability to process projects with varying complexity.

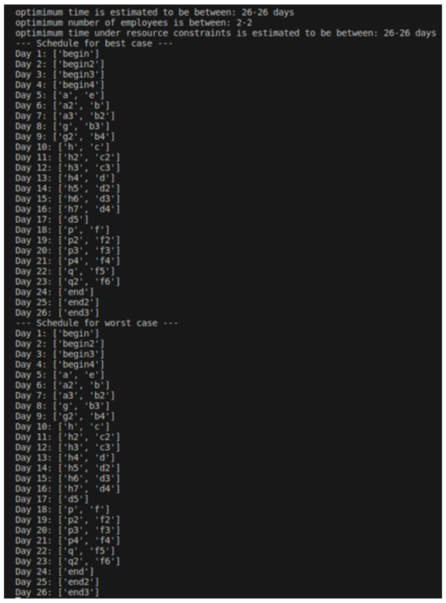
Each test involved manually inspecting the results, comparing them to expected schedules, and adjusting inputs where necessary to validate the accuracy and robustness of the system. Edge cases were also tested to evaluate how the platform handles unusual or problematic scenarios.

5.2 Functional Testing

Fixed durations

Figure 4.1 illustrates a sample project graph used for testing. Assuming two employees are available and that all task durations are fixed (i.e., min and max estimates are equal), the generated schedule is shown in Figure 5.1.

The results indicate that the system correctly determines the optimum project duration, which aligns with the expected theoretical output. Since the calculated optimum number of employees is also 2—matching the available workforce—the resource constraints do not introduce any delays. Additionally, each day's schedule contains at most two concurrent tasks, confirming



```
optimimum time is estimated to be between: 26-26 days
optimimum number of employees is between: 2-2
optimimum time under resource constraints is estimated to be between: 26-26 days
--- Schedule for best case ---
Day 1: ['begin']
Day 2: ['begin2']
Day 3: ['begin3']
Day 4: ['begin4']
Day 5: ['a', 'e']
Day 6: ['a2', 'b']
Day 7: ['a3', 'b2']
Day 8: ['g', 'b3']
Day 9: ['g2', 'b4']
Day 10: ['h', 'c']
Day 11: ['h2', 'c2']
Day 12: ['h3', 'c3']
Day 13: ['h4', 'd']
Day 14: ['h5', 'd2']
Day 15: ['h6', 'd3']
Day 16: ['h7', 'd4']
Day 17: ['d5']
Day 18: ['p', 'f']
Day 19: ['p2', 'f2']
Day 20: ['p3', 'f3']
Day 21: ['p4', 'f4']
Day 22: ['q', 'f5']
Day 23: ['q2', 'f6']
Day 24: ['end']
Day 25: ['end2']
Day 26: ['end3']
--- Schedule for worst case ---
Day 1: ['begin']
Day 2: ['begin2']
Day 3: ['begin3']
Day 4: ['begin4']
Day 5: ['a', 'e']
Day 6: ['a2', 'b']
Day 7: ['a3', 'b2']
Day 8: ['g', 'b3']
Day 9: ['g2', 'b4']
Day 10: ['h', 'c']
Day 11: ['h2', 'c2']
Day 12: ['h3', 'c3']
Day 13: ['h4', 'd']
Day 14: ['h5', 'd2']
Day 15: ['h6', 'd3']
Day 16: ['h7', 'd4']
Day 17: ['d5']
Day 18: ['p', 'f']
Day 19: ['p2', 'f2']
Day 20: ['p3', 'f3']
Day 21: ['p4', 'f4']
Day 22: ['q', 'f5']
Day 23: ['q2', 'f6']
Day 24: ['end']
Day 25: ['end2']
Day 26: ['end3']
```

Figure 5.1: Scheduler output for fixed durations

that resource limits are respected. The output also ensures that no task is scheduled before its dependencies are completed, verifying that task dependencies are correctly enforced.

Variable Durations

To evaluate how the system handles uncertainty in task durations, the maximum estimates were increased while keeping the minimum estimates unchanged. The resulting worst-case project graph is depicted in Figure 5.2, and the corresponding worst-case schedule is presented in Figure 5.3.

With these updated estimates, the projected completion time range expands from 26 days to 26–48 days. This adjustment accounts for the increased duration of certain tasks, demonstrating that the algorithm successfully adapts to worst-case scenarios. The best-case schedule remains unchanged, showing that the system correctly handles varying task duration estimates by maintaining separate schedules for best and worst cases.

Suboptimal Resources

Previous test cases were conducted with the number of employees set to the calculated optimum. To evaluate how the system adapts to resource limitations, the tests were repeated with only one available employee. This resulted in the schedules shown in Figure 5.4 for the best-case scenario and Figure 5.5 for the worst-case scenario.

With only one employee available, the schedule is adjusted to ensure that no more than

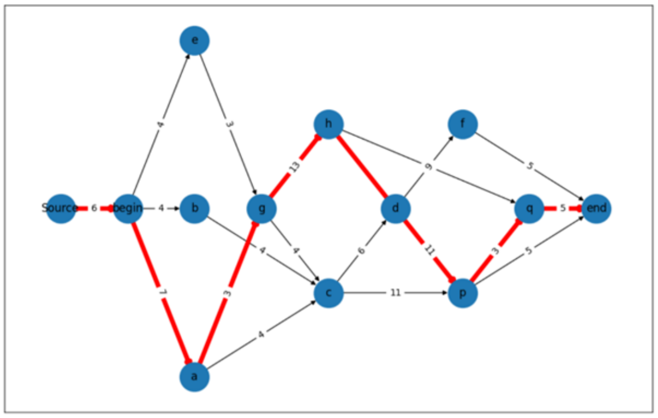
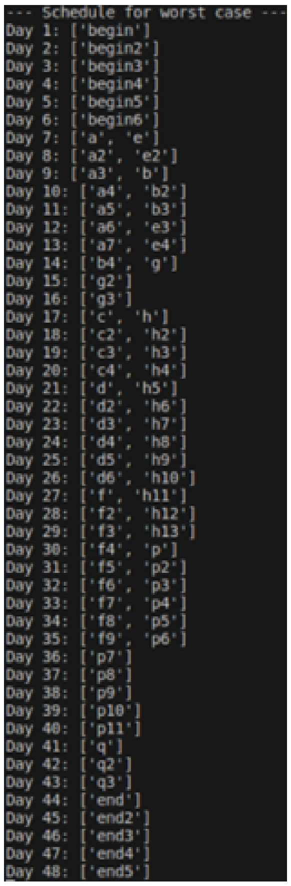
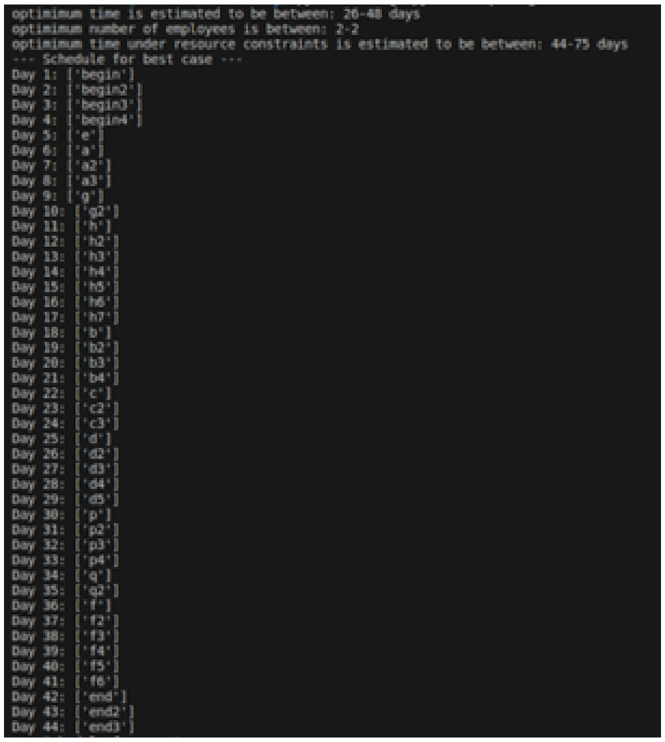


Figure 5.2: Graph output for worse case durations



```
--- Schedule for worst case ---
Day 1: ['begin']
Day 2: ['begin2']
Day 3: ['begin3']
Day 4: ['begin4']
Day 5: ['begin5']
Day 6: ['begin6']
Day 7: ['a', 'e']
Day 8: ['a2', 'e2']
Day 9: ['a3', 'b']
Day 10: ['a4', 'b2']
Day 11: ['a5', 'b3']
Day 12: ['a6', 'e3']
Day 13: ['a7', 'e4']
Day 14: ['b4', 'g']
Day 15: ['g2']
Day 16: ['g3']
Day 17: ['c', 'h']
Day 18: ['c2', 'h2']
Day 19: ['c3', 'h3']
Day 20: ['c4', 'h4']
Day 21: ['d', 'h5']
Day 22: ['d2', 'h6']
Day 23: ['d3', 'h7']
Day 24: ['d4', 'h8']
Day 25: ['d5', 'h9']
Day 26: ['d6', 'h10']
Day 27: ['f', 'h11']
Day 28: ['f2', 'h12']
Day 29: ['f3', 'h13']
Day 30: ['f4', 'p']
Day 31: ['f5', 'p2']
Day 32: ['f6', 'p3']
Day 33: ['f7', 'p4']
Day 34: ['f8', 'p5']
Day 35: ['f9', 'p6']
Day 36: ['p7']
Day 37: ['p8']
Day 38: ['p9']
Day 39: ['p10']
Day 40: ['p11']
Day 41: ['q']
Day 42: ['q2']
Day 43: ['q3']
Day 44: ['end']
Day 45: ['end2']
Day 46: ['end3']
Day 47: ['end4']
Day 48: ['end5']
```

Figure 5.3: Schedule output for worse case durations



```
optiminum time is estimated to be between: 26-48 days
optiminum number of employees is between: 2-2
optiminum time under resource constraints is estimated to be between: 44-75 days
--- Schedule for best case ---
Day 1: ['begin']
Day 2: ['begin2']
Day 3: ['begin3']
Day 4: ['begin4']
Day 5: ['e']
Day 6: ['a']
Day 7: ['a2']
Day 8: ['a3']
Day 9: ['g']
Day 10: ['g2']
Day 11: ['h']
Day 12: ['h2']
Day 13: ['h3']
Day 14: ['h4']
Day 15: ['h5']
Day 16: ['h6']
Day 17: ['h7']
Day 18: ['b']
Day 19: ['b2']
Day 20: ['b3']
Day 21: ['b4']
Day 22: ['c']
Day 23: ['c2']
Day 24: ['c3']
Day 25: ['d']
Day 26: ['d2']
Day 27: ['d3']
Day 28: ['d4']
Day 29: ['d5']
Day 30: ['p']
Day 31: ['p2']
Day 32: ['p3']
Day 33: ['p4']
Day 34: ['q']
Day 35: ['q2']
Day 36: ['f']
Day 37: ['f2']
Day 38: ['f3']
Day 39: ['f4']
Day 40: ['f5']
Day 41: ['f6']
Day 42: ['end']
Day 43: ['end2']
Day 44: ['end3']
```

Figure 5.4: Schedule output for best case under 1 employee

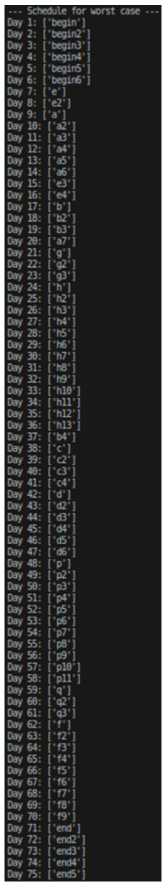
one task is executed per day. As a result, the overall project completion time increases from 26 to 44 days in the best case and from 48 to 75 days in the worst case. This demonstrates how resource constraints directly impact project duration and highlights the system's ability to dynamically restructure the schedule while still maintaining task dependencies.

Testing Mutual Exclusions

Referring back to the example in Figure 5.1, tasks A and B are scheduled to run simultaneously on days 6 and 7. If a mutual exclusion constraint is introduced, the system should adjust the schedule to ensure that these tasks are no longer executed on the same day.

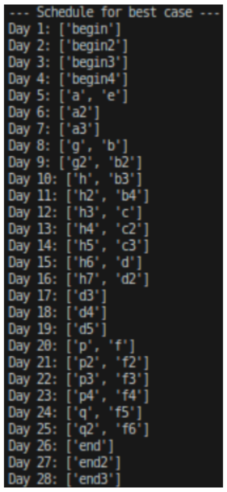
To verify this, the database was updated to enforce the mutual exclusion, and the algorithm was rerun. The resulting schedule, shown in Figure 5.6, confirms that tasks A and B are now scheduled separately. This demonstrates that the system correctly handles mutual exclusions by restructuring the schedule accordingly.

To further evaluate the handling of mutual exclusions, we conducted another test using two tasks that were originally scheduled on different days and did not overlap. For this test, tasks C and G were selected. After updating the database to enforce a mutual exclusion between them and rerunning the system, the resulting schedule is shown in Figure 5.7. As expected, the



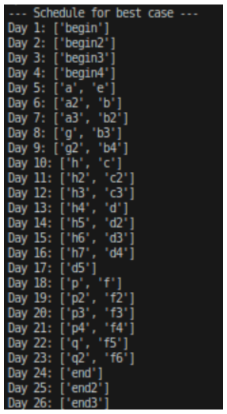
```
--- Schedule for worst case ---
Day 1: ['begin']
Day 2: ['begin2']
Day 3: ['begin3']
Day 4: ['begin4']
Day 5: ['begin5']
Day 6: ['begin6']
Day 7: ['e']
Day 8: ['e2']
Day 9: ['a']
Day 10: ['a2']
Day 11: ['a3']
Day 12: ['a4']
Day 13: ['a5']
Day 14: ['a6']
Day 15: ['e3']
Day 16: ['e4']
Day 17: ['b']
Day 18: ['b2']
Day 19: ['b3']
Day 20: ['a7']
Day 21: ['g']
Day 22: ['g2']
Day 23: ['g3']
Day 24: ['h']
Day 25: ['h2']
Day 26: ['h3']
Day 27: ['h4']
Day 28: ['h5']
Day 29: ['h6']
Day 30: ['h7']
Day 31: ['h8']
Day 32: ['h9']
Day 33: ['h10']
Day 34: ['h11']
Day 35: ['h12']
Day 36: ['h13']
Day 37: ['b4']
Day 38: ['c']
Day 39: ['c2']
Day 40: ['c3']
Day 41: ['c4']
Day 42: ['d']
Day 43: ['d2']
Day 44: ['d3']
Day 45: ['d4']
Day 46: ['d5']
Day 47: ['d6']
Day 48: ['p']
Day 49: ['p2']
Day 50: ['p3']
Day 51: ['p4']
Day 52: ['p5']
Day 53: ['p6']
Day 54: ['p7']
Day 55: ['p8']
Day 56: ['p9']
Day 57: ['p10']
Day 58: ['p11']
Day 59: ['q']
Day 60: ['q2']
Day 61: ['q3']
Day 62: ['f']
Day 63: ['f2']
Day 64: ['f3']
Day 65: ['f4']
Day 66: ['f5']
Day 67: ['f6']
Day 68: ['f7']
Day 69: ['f8']
Day 70: ['f9']
Day 71: ['end']
Day 72: ['end2']
Day 73: ['end3']
Day 74: ['end4']
Day 75: ['end5']
```

Figure 5.5: Schedule output for worst case under 1 employee



```
--- Schedule for best case ---
Day 1: ['begin']
Day 2: ['begin2']
Day 3: ['begin3']
Day 4: ['begin4']
Day 5: ['a', 'e']
Day 6: ['a2']
Day 7: ['a3']
Day 8: ['g', 'b']
Day 9: ['g2', 'b2']
Day 10: ['h', 'b3']
Day 11: ['h2', 'b4']
Day 12: ['h3', 'c']
Day 13: ['h4', 'c2']
Day 14: ['h5', 'c3']
Day 15: ['h6', 'd']
Day 16: ['h7', 'd2']
Day 17: ['d3']
Day 18: ['d4']
Day 19: ['d5']
Day 20: ['p', 'f']
Day 21: ['p2', 'f2']
Day 22: ['p3', 'f3']
Day 23: ['p4', 'f4']
Day 24: ['q', 'f5']
Day 25: ['q2', 'f6']
Day 26: ['end']
Day 27: ['end2']
Day 28: ['end3']
```

Figure 5.6: Schedule output for case when A and B are mutually exclusive



```
--- Schedule for best case ---
Day 1: ['begin']
Day 2: ['begin2']
Day 3: ['begin3']
Day 4: ['begin4']
Day 5: ['a', 'e']
Day 6: ['a2', 'b']
Day 7: ['a3', 'b2']
Day 8: ['g', 'b3']
Day 9: ['g2', 'b4']
Day 10: ['h', 'c']
Day 11: ['h2', 'c2']
Day 12: ['h3', 'c3']
Day 13: ['h4', 'd']
Day 14: ['h5', 'd2']
Day 15: ['h6', 'd3']
Day 16: ['h7', 'd4']
Day 17: ['d5']
Day 18: ['p', 'f']
Day 19: ['p2', 'f2']
Day 20: ['p3', 'f3']
Day 21: ['p4', 'f4']
Day 22: ['q', 'f5']
Day 23: ['q2', 'f6']
Day 24: ['end']
Day 25: ['end2']
Day 26: ['end3']
```

Figure 5.7: Schedule output for case when C and G are mutually exclusive

schedule remains unchanged, confirming that the system correctly applies mutual exclusions only when tasks were initially scheduled to run concurrently.

Variable Employee Requirements

In some scenarios, the required number of employees can vary between the best and worst cases. In such situations, the system must be able to accurately identify the range of employees needed, rather than relying on a fixed number. This is illustrated in Figures 5.8 and 5.9, which depict the best and worst case scenarios for a specific project. In the case where two employees are available, the longest path in the minimum case graph remains unchanged. However, in the maximum case graph, as shown in Figure 5.10, adjustments are necessary. To ensure that no more than two tasks exist per topological layer, the longest path must be extended from 4 to 5. This modification is accurately reflected in the schedule output, as demonstrated in Figure 5.11.

5.3 System Evaluation

Scalability

To evaluate the scalability of the scheduling platform, test cases were conducted with different numbers of tasks: 100+, 500+, and 1000+. The execution times for generating the schedules were measured as approximately 2 seconds, 33 seconds, and 1 minute 42 seconds, respectively.

From these results, the execution time does not scale linearly with the number of tasks but instead exhibits a superlinear growth pattern. This suggests that certain operations within the scheduling process contribute more than just the longest path algorithm's theoretical O(V +

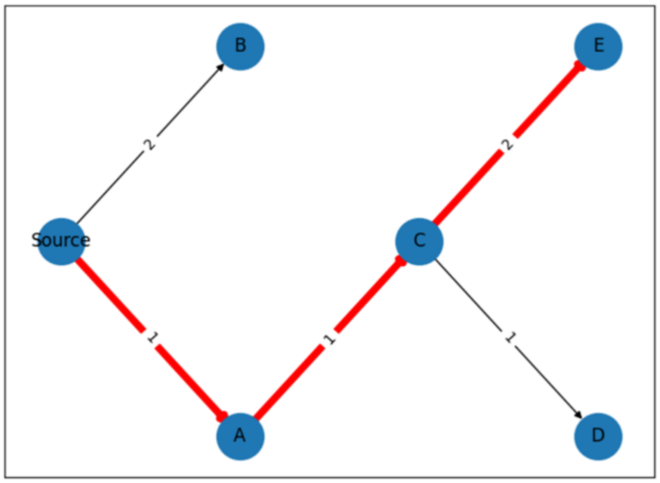


Figure 5.8: Min estimate graph

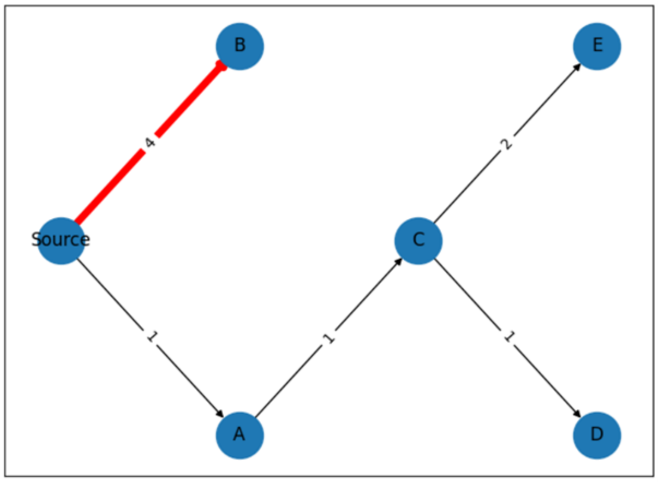


Figure 5.9: Max estimate graph

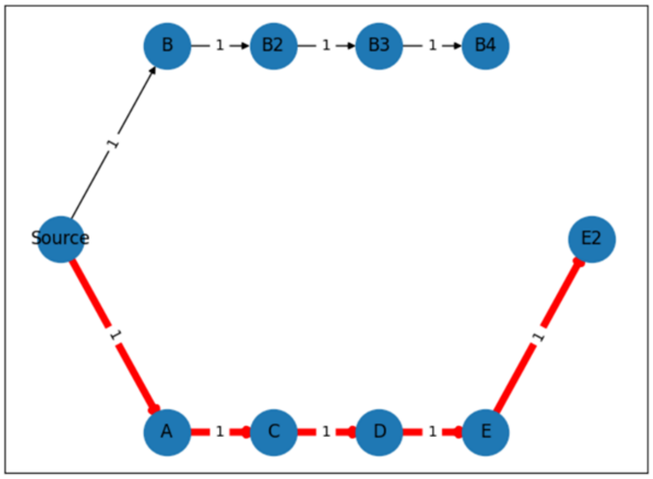
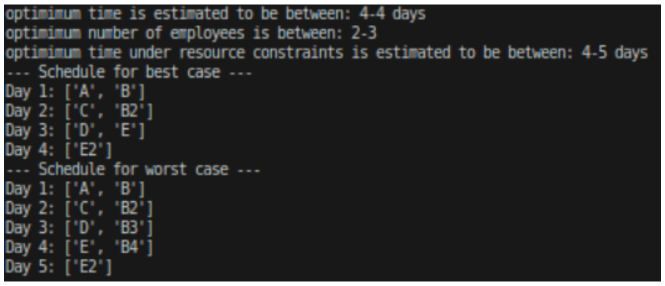


Figure 5.10: graph showing the longest path in the worst case with 2 employees



```
optimimum time is estimated to be between: 4-4 days
optimimum number of employees is between: 2-3
optimimum time under resource constraints is estimated to be between: 4-5 days
--- Schedule for best case ---
Day 1: ['A', 'B']
Day 2: ['C', 'B2']
Day 3: ['D', 'E']
Day 4: ['E2']
--- Schedule for worst case ---
Day 1: ['A', 'B']
Day 2: ['C', 'B2']
Day 3: ['D', 'B3']
Day 4: ['E', 'B4']
Day 5: ['E2']
```

Figure 5.11: Schedule output with variable employee requirements

E) complexity.

Several factors could explain this trend. First, the overhead from constructing the graph plays a role, as each task must be retrieved from the database and processed into a directed acyclic graph (DAG). The time complexity of database queries and graph-building operations can significantly impact performance. Additionally, modifications for handling mutual exclusions and resource constraints introduce extra dependencies, making the process more complex than the base longest path calculation. The conversion to a uniform graph further contributes to the execution time, as tasks with durations greater than one are split into multiple smaller nodes, effectively increasing the number of vertices being processed.

If this trend continues, larger project sizes, such as those with 5000+ tasks, may lead to execution times that are impractical for real-time scheduling.

Heuristic Approach

One drawback of the adapted scheduling method compared to the standard longest path algorithm is that it introduces a heuristic approach rather than providing a guaranteed optimal solution. The standard longest path algorithm operates under the assumption that all tasks without dependencies can be executed concurrently, given unlimited resources. This allows it to deterministically compute the shortest possible project completion time.

With the adaptation of limited resources and mutual exclusions, the system does not explore all possible configurations. Instead, they follow heuristic rules to approximate an optimal schedule. For example, in the case of resource allocation, the approach prioritises balancing the number of concurrent tasks rather than exhaustively testing all possible orderings. This means that while the system ensures feasibility and improves upon a naive scheduling method, it does not guarantee finding the absolute best possible schedule in all cases.

This trade-off is necessary due to computational feasibility. A brute-force method that evaluates all possible task orderings and resource allocations would grow exponentially, making it infeasible for large-scale projects. Instead, the heuristic-based approach provides a reasonable balance between efficiency and accuracy, offering a practical solution that can handle large datasets while still producing schedules that adhere to project constraints. However, users should be aware that the generated schedules may not always be globally optimal and may, in some cases, leave room for further manual refinement by project managers.

Usability

Another drawback of the current implementation is the lack of a graphical user interface (GUI), which makes interacting with the platform less intuitive. At present, users must manually modify the database to input project details, including tasks, dependencies, exclusions, and resource constraints. This process is not only time-consuming but also prone to human error, as incorrect database entries could lead to inaccurate scheduling results. A user-friendly GUI would streamline this process, allowing project managers to input data through forms and dropdown menus rather than directly modifying database tables.

Additionally, while the platform generates a visual representation of the project schedule, this becomes increasingly impractical as the number of tasks grows. For small projects, the graph representation is useful in illustrating task dependencies and the longest path. However, for large-scale projects with hundreds or thousands of tasks, the visualisation quickly becomes unreadable due to the density of nodes and edges. The cluttered graph makes it difficult for users to extract meaningful insights at a glance.

Future improvements could address these limitations by integrating an interactive GUI that allows users to input and edit project data more efficiently. Dynamic filtering and zooming mechanisms could also be implemented for graph visualisation, enabling users to focus on specific task sequences or critical paths while hiding less relevant details. Another potential enhancement is the inclusion of alternative views, such as Gantt charts, to present scheduling information in a more digestible format. These improvements would make the platform more accessible to a wider range of users while enhancing the clarity of the generated schedules.

Chapter 6

Legal, Social, Ethical and Professional Issues

6.1 Brief

When developing a project scheduling and resource management platform, it is essential to consider the legal, social, ethical, and professional implications of its implementation. Since the system is designed to assist in task scheduling, dependency management, and resource allocation, it interacts with critical aspects of project management that can have real-world consequences on employees, businesses, and stakeholders.

6.2 Legal Considerations

A core legal consideration for this project is data protection and privacy. Although the system does not store sensitive personal information, it does involve project data that may include employee task assignments and scheduling details. If deployed in an organisational setting, it must comply with the UK General Data Protection Regulation (UK GDPR) [6] and Data Protection Act 2018 [7]. Organisations using the system must ensure that employee data is stored securely, access is restricted to authorised personnel, and that data is used only for its intended purpose.

Additionally, if this platform were to be commercialised, compliance with intellectual property laws would be necessary. Any third-party libraries, including NetworkX (used for graph operations) and Django (used for the backend framework), must be used in accordance with their open-source licences. Attribution, licensing compatibility, and proper documentation must be maintained to avoid any potential legal disputes.

Furthermore, as the system provides recommendations on task scheduling, it is important to consider employment laws to ensure that the generated schedules comply with working hours regulations and do not inadvertently promote excessive workloads or violate fair labour standards.

6.3 Social & Ethical Considerations

The platform has the potential to improve efficiency in project management, helping organisations optimize their resources and minimize project delays. However, improper use of the system could lead to unintended consequences, such as workforce exploitation. If employers rely too heavily on algorithmic scheduling without considering employee well-being, workers may be pressured to take on excessive workloads, leading to burnout and job dissatisfaction.

Another concern is bias and fairness in scheduling decisions. The algorithm prioritizes efficiency but does not inherently account for individual employee preferences, skills, or work-life balance. If used improperly, it could result in some employees consistently receiving less favourable schedules. Ensuring fairness requires incorporating flexibility, such as allowing manual adjustments and considering employee availability and preferences where possible.

Chapter 7

Conclusion and Future Work

This project has demonstrated the value of algorithmic solutions in optimizing task scheduling within complex projects. Through the development and implementation of an adapted longest path algorithm, the system effectively addresses critical challenges in modern project management, such as uncertain task durations, mutually exclusive tasks, and resource management. Additionally, it provides insights into improving project timelines, showcasing the system's potential for real-world applications.

This project also lays a solid foundation for future work, including optimising the algorithm further to handle larger projects, integrating real-time data, and developing a user-friendly interface to enhance its accessibility. Further development of the system could see it evolve into a marketable product, offering significant value to industries where complex project scheduling and resource management are essential.

References

[1] Monique Aubry. Project management office transformations: Direct and moderating effects that enhance performance and maturity. *Project Management Journal*, 46(5):19–45, 2015.

[2] https://www.oracle.com/uk/construction-engineering/primavera-p6/ accessed 31 March 2025.

[3] https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pulse/pulse-of-the-profession-2017.pdf.

[4] Tomasz Radzik. 6ccs3ome/7ccsmome – optimisation methods lecture 2.

[5] Daniel Alejandro Rossit, Fernando Tohmé, and Mariano Frutos. Industry 4.0: smart scheduling. *International Journal of Production Research*, 57(12):3802–3813, 2019.

[6] UK Government. UK General Data Protection Regulation (UK GDPR), 2016. Accessed: April 3, 2025.

[7] UK Government. Data Protection Act 2018, 2018. Accessed: April 3, 2025.