

People's Democratic Republic of Algeria  
Ministry of Higher Education and Scientific Research  
University of Science and Technology Houari Boumediene



Faculty of Mathematics  
Department of Operational Research

## Final Study Project

Licence degree in  
Operational Research

---

# Task Scheduling Problem

---

Presented by:

KHOUTRI IMAD

ABDOU MOUBAREK

Supervised by:

CHAIBLAINE Yacine

MAY 2025

# Acknowledgments

**Alhamdulillah**, all praise is due to Allah, the Almighty, for granting us the strength, patience, and perseverance throughout this academic journey. Without His guidance and blessings, this work would not have been possible.

We express our sincere gratitude to our supervisor, **Dr. Yacine Chaiblaine**, for his invaluable guidance, insightful advice, and patience throughout this project. His expertise and support have been of great help, and we have learned so much under his mentorship.

Our appreciation extends to all the professors who have contributed, directly or indirectly, to our academic journey. Each of them has left a mark on our path, and we are deeply grateful for their dedication and knowledge-sharing.

Finally, a heartfelt thank you to everyone who has supported us in any way during this journey. Whether through words of encouragement, intellectual discussions, or simply their kind presence, their support has meant a lot to us.

**Hamdoulillah** for this experience, the lessons learned, and the progress made.

# Dedication

To my dear **Mother**, whose unconditional love and support have been a source of strength and inspiration.

To my **Father** for his sacrifices and his support, which have given me the opportunity to pursue my dreams with confidence.

To my **Friends** for their genuine friendship and support throughout this journey.

To my roommate **Abdelsamed** for the moments of sharing and support that made this experience even more meaningful.

To **N. Boumediene**, whose presence has been a guiding **light** in my life. Your kindness, encouragement, and belief in me have been a source of strength in ways that words cannot fully express, I am endlessly grateful for the **light** you bring into my days.

To my partner **ABDOU Moubarek**, with whom I have shared this academic adventure, filled with challenges and achievements.

To all my friends, family, and everyone who has contributed, in some way or another, to my growth and the success of my academic journey.

**Thank you all, from the bottom of my heart.**

**Imad**

# Dedication

To my **my Parents**, For your endless love, constant support, and encouragement, which kept me going throughout my studies.

To my **Brothers and Sisters**, For your warm presence and encouragement, which have accompanied me with kindness at every step of this endeavor.

To my **Cousins and Friends**, For your kindness, support, and happy moments that made my journey brighter.

To my partner, **KHOUTRI Imad**, For your dedication, teamwork, and support, which gave me strength during this academic journey.

To my **Roommate**, For our meaningful conversations, mutual help, and shared memories that made these years unforgettable

Moubarek

# Contents

Résumé	7
8	المخلص
Abstract	9
Introduction	10
Chapter 1: Fundamental Definitions	11
1 Task Scheduling	11
2 Operations Research	11
3 Optimization	12
3.1 Decision Variables . . . . .	12
3.2 Objective function . . . . .	12
3.3 Constraints . . . . .	12
4 Scheduling Types and Approaches	12
4.1 Static vs. Dynamic Scheduling . . . . .	12
4.1.1 Static . . . . .	12
4.1.2 Dynamic . . . . .	13
4.2 Deterministic vs. Stochastic Scheduling . . . . .	13
4.2.1 Deterministic . . . . .	13
4.2.2 Stochastic . . . . .	13
4.3 Offline vs. Online Scheduling . . . . .	13
4.3.1 Offline Scheduling . . . . .	14
4.3.2 Online Scheduling . . . . .	14
4.4 Single-Machine vs. Multi-Machine Scheduling . . . . .	14
4.4.1 Single Machine . . . . .	14
4.4.2 Multi-Machine . . . . .	14
4.5 Flow Shop and Job Shop Scheduling . . . . .	14
4.5.1 Flow Shop . . . . .	14
4.5.2 Job Shop . . . . .	15
5 Real-World Applications of Task Scheduling	15
5.1 Manufacturing and Production Planning . . . . .	15
5.2 Healthcare Systems . . . . .	16
5.3 Cloud Computing and IT Services . . . . .	16
5.4 Transportation and Logistics . . . . .	17
6 Mathematical and Algorithmic Foundations	17
6.1 Graph theory . . . . .	17
6.2 Complexity Theory . . . . .	18
6.2.1 P Problems . . . . .	18
6.2.2 NP Problems . . . . .	19

6.2.3	NP-Completeness . . . . .	19
6.2.4	NP-Hard Problems . . . . .	19
6.3	Heuristics and Metaheuristics . . . . .	20
6.3.1	Heuristics . . . . .	20
6.3.2	Metaheuristics . . . . .	20
<b>Chapter 2: Methodologies and Algorithms</b>		<b>22</b>
<b>7</b>	<b>Difference Between Exact, Heuristic, and Metaheuristic Approaches</b>	<b>22</b>
<b>8</b>	<b>Exact algorithms</b>	<b>23</b>
8.1	Linear Programming (LP) and Integer Programming (IP) . . . . .	23
8.2	Branch and Bound (B&B) . . . . .	24
8.3	Dynamic Programming (DP) . . . . .	25
<b>9</b>	<b>Heuristic Approaches</b>	<b>25</b>
9.1	List Scheduling . . . . .	25
9.2	Greedy Algorithms . . . . .	26
9.3	Priority-based Scheduling . . . . .	27
<b>10</b>	<b>Metaheuristic Approaches</b>	<b>28</b>
10.1	Genetic Algorithms (GA) . . . . .	28
10.2	Simulated Annealing (SA) . . . . .	29
10.3	Ant Colony Optimization (ACO) . . . . .	29
10.4	Particle Swarm Optimization (PSO) . . . . .	30
10.5	Tabu Search (TS) . . . . .	30
<b>11</b>	<b>Performance and Evaluation Metrics</b>	<b>31</b>
11.1	Makespan . . . . .	31
11.2	Throughput . . . . .	31
11.3	Response Time . . . . .	31
11.4	Latency . . . . .	32
11.5	Load Balancing . . . . .	32
11.6	Fairness . . . . .	32
<b>Chapter 3: Results and Discussion</b>		<b>34</b>
<b>12</b>	<b>Problem Definition</b>	<b>34</b>
12.1	Problem Parameters . . . . .	34
12.2	Constraints . . . . .	34
12.3	Objective Function . . . . .	35
<b>13</b>	<b>Exact Approach: Integer Linear Programming</b>	<b>35</b>
13.1	Mathematical Formulation . . . . .	35
13.1.1	Decision Variables . . . . .	35
13.1.2	Objective Function . . . . .	35
13.1.3	Constraints . . . . .	35
13.2	Python Implementation . . . . .	36
13.3	Results and Analysis . . . . .	40

<b>14 Analysis of the Exact Approach</b>	<b>41</b>
14.1 Strengths and Limitations of Integer Linear Programming . . . . .	41
<b>Chapter 4: Task Scheduling Application</b>	<b>43</b>
<b>15 Introduction to the Application</b>	<b>43</b>
<b>16 Theoretical Foundation</b>	<b>43</b>
<b>17 Application Architecture and Design</b>	<b>44</b>
17.1 Core Components . . . . .	44
17.2 User Interface Design . . . . .	44
17.3 Data Model . . . . .	44
<b>18 Implementation of the Exact Method (PuLP)</b>	<b>45</b>
<b>19 Implementation of the Metaheuristic Method (Simulated Annealing)</b>	<b>45</b>
<b>20 User Interface and Experience</b>	<b>45</b>
20.1 Task Management Interface . . . . .	45
<b>21 Method Selection in the Application</b>	<b>46</b>
21.1 Choosing Between Exact and Metaheuristic Methods . . . . .	46
21.2 Resource Management Interface . . . . .	47
21.3 Schedule Visualization . . . . .	48
<b>22 Application Features</b>	<b>49</b>
22.1 Task Management . . . . .	49
22.2 Resource Management . . . . .	50
22.3 Schedule Generation . . . . .	50
22.4 Results Visualization and Export . . . . .	50
<b>23 Mobile Responsiveness and User Experience</b>	<b>51</b>
23.1 Responsive Design Philosophy . . . . .	52
23.2 Mobile Timeline View . . . . .	52
23.3 Mobile Task List View . . . . .	53
23.4 Mobile Resource Management . . . . .	54
<b>24 Case Study and Results</b>	<b>54</b>
24.1 Problem Description . . . . .	54
24.2 Results Comparison . . . . .	55
24.3 Analysis of Results . . . . .	55
<b>25 Final thoughts</b>	<b>56</b>
<b>Conclusion</b>	<b>57</b>

# Résumé

La planification des tâches est essentielle dans de nombreux domaines, allant des opérations industrielles à la gestion de projets en passant par l'optimisation des services. Malgré son importance, de nombreuses structures — notamment en Algérie — continuent d'utiliser des méthodes manuelles ou inefficaces, causant des retards, une perte de ressources et une baisse de productivité. Dans ce travail, nous abordons le problème de la planification des tâches à travers des méthodes exactes, heuristiques, et meta-heuristiques en combinant une base théorique solide à une implémentation pratique. Le cœur de notre contribution est une application conviviale conçue pour aider les individus et les entreprises à organiser et optimiser leurs plannings de manière efficace. Cette application comble un vide réel sur le marché, en proposant un outil accessible et personnalisable, adapté à divers scénarios. Face à une demande croissante pour une meilleure gestion du temps et des ressources, cette solution est non seulement pertinente mais aussi indispensable — notamment dans les économies émergentes où les outils numériques de planification restent rares.

Notre travail constitue ainsi une base prometteuse pour des améliorations futures et une diffusion à plus grande échelle.



## الملخص

تُعدّ جدولة المهام عنصراً أساسياً في العديد من المجالات، من العمليات الصناعية إلى إدارة المشاريع وتحسين الخدمات. ورغم أهميتها، لا تزال العديد من المؤسسات خاصة في الجزائر تعتمد على طرق تقليدية أو غير فعّالة، مما يؤدي إلى تأخيرات وهدر في الموارد وانخفاض في الإنتاجية. في هذا العمل، قننا بمعالجة مشكلة جدولة المهام باستخدام طرق دقيقة وأخرى تقريبية (استكشافية)، مع الجمع بين أساس نظري قوي وتطبيق عملي.

يتمثل الإنجاز الرئيسي في هذا المشروع في تطوير تطبيق سهل الاستخدام يُمكن الأفراد والمؤسسات من تنظيم وجدولة مهامهم بشكل فعّال. هذا التطبيق يملأ فجوة حقيقية في السوق، حيث يوفر أداة مرنة وسهلة التكيف مع مختلف الحالات. ومع تزايد الحاجة إلى إدارة أفضل للوقت والموارد، تُعدّ هذه الحلول ضرورية وملحّة، خاصة في الدول الناشئة التي تفتقر إلى أدوات رقمية فعّالة في هذا المجال. يشكّل هذا المشروع خطوة أولى نحو تطوير أدوات أكثر تطوراً ونشرها على نطاق واسع.

# Abstract

Task scheduling is a crucial component in numerous domains, from industrial operations to project management and service optimization. Despite its importance, many organizations—especially in Algeria—still rely on manual or inefficient methods that lead to delays, resource wastage, and reduced productivity. In this work, we tackle the task scheduling problem through both exact heuristic, meta-heuristic methods, providing a theoretical foundation enriched by practical implementation. The centerpiece of our contribution is a user-friendly application designed to assist individuals and businesses in organizing and optimizing their task schedules efficiently. Our app addresses a real gap in the market, offering an accessible and customizable tool adaptable to a wide range of scenarios. With growing demands for better time and resource management across the world, this solution is not only relevant but essential—especially in emerging economies where digital scheduling tools remain scarce. Our work provides a solid base for future enhancements and large-scale deployment.

# Introduction

Task scheduling represents one of the most fundamental and challenging problems in operations research and computer science. At its core, it involves the allocation of limited resources to tasks over time, with the goal of optimizing one or more performance criteria. This seemingly straightforward concept belies the immense complexity that emerges when dealing with real-world constraints, dependencies, and objectives.

In today's fast-paced and resource-constrained environment, efficient task scheduling has become increasingly critical across diverse domains—from manufacturing and production planning to software development, healthcare services, and cloud computing. Organizations constantly seek to maximize productivity, minimize costs, and deliver products or services within tight deadlines. Effective scheduling directly impacts these goals by ensuring optimal resource utilization, reducing idle time, and minimizing overall completion time.

The significance of task scheduling extends beyond operational efficiency. In manufacturing, it can reduce energy consumption and environmental impact. In healthcare, it can improve patient care and reduce waiting times. In software development, it can accelerate time-to-market and enhance team collaboration. The ubiquity of scheduling challenges across industries underscores the importance of developing robust, flexible, and efficient scheduling methodologies.

This project explores the multifaceted nature of task scheduling problems, examining both theoretical foundations and practical applications. We begin by establishing fundamental definitions and concepts that form the basis of scheduling theory. We then delve into various solution methodologies, ranging from exact mathematical approaches to heuristic and metaheuristic techniques. Through detailed analysis and implementation, we demonstrate how these different approaches can be applied to solve complex scheduling problems.

The culmination of our work is the development of a comprehensive task scheduling application that implements both exact methods (using PuLP for Integer Linear Programming) and metaheuristic approaches (using Simulated Annealing). This application serves as a practical tool for generating optimized schedules while providing insights into the strengths and limitations of different scheduling techniques.

Throughout this project, we maintain a balance between theoretical rigor and practical applicability. While we explore the mathematical underpinnings of scheduling algorithms, we also focus on their implementation in real-world scenarios. This dual perspective allows us to bridge the gap between abstract models and concrete solutions, providing valuable insights for both researchers and practitioners in the field.

# Chapter 1: Fundamental Definitions

This chapter provides essential definitions and fundamental concepts necessary for understanding the core idea of this project. Focusing on task scheduling in a general sense, this chapter establishes a foundation that spans multiple domains—ranging from manufacturing to computing and service industries. By defining key terms and introducing core methodologies, we lay the groundwork for the next chapters that will delve into detailed methodologies and practical discussions.

## 1 Task Scheduling

Scheduling is a structured decision-making process that is widely utilized in manufacturing, computing, and service industries. It involves the allocation of available resources to specific tasks over defined time periods, with the objective of optimizing one or more performance criteria [1].

To build a comprehensive understanding, we now extend our discussion to the broader analytical framework that supports scheduling decisions.

## 2 Operations Research

Operations Research (O.R.) is a discipline that deals with the application of advanced analytical methods to help make better decisions.

Employing techniques from other mathematical sciences, such as mathematical modeling, statistical analysis, and mathematical optimization, operations research arrives at optimal or near-optimal solutions to complex decision-making problems.

Operations research overlaps with other disciplines, notably industrial engineering and operations management. It is often concerned with determining a maximum (such as profit, performance, or yield) or minimum (such as loss, risk, or cost).

Operations research encompasses a wide range of problem-solving techniques and methods applied in the pursuit of improved decision-making and efficiency, such as simulation, mathematical optimization, queuing theory, Markov decision processes, economic methods, data analysis, statistics, neural networks, expert systems, and decision analysis. Nearly all of these techniques involve the construction of mathematical models that attempt to describe the system [2].

With the decision-making context established through operations research, we now turn to a more specific area: **Optimization**

## 3 Optimization

Optimization is the process of finding the best possible solution (maximum or minimum) to a problem within a given set of constraints. It involves selecting the optimal values of decision variables to achieve the desired objective, such as maximizing profit, minimizing cost, or improving efficiency [3].

### 3.1 Decision Variables

Decision variables are the unknown quantities in an optimization problem that need to be determined. They represent the choices or decisions that can be adjusted to achieve the best outcome, such as the number of products to produce or the amount of resources to allocate.

### 3.2 Objective function

The objective function is a mathematical expression that defines the goal of an optimization problem. It represents the quantity to be maximized (e.g., profit, efficiency) or minimized (e.g., cost, waste) based on the decision variables.

### 3.3 Constraints

Constraints are the limitations or restrictions imposed on an optimization problem. They define the feasible region by specifying conditions that the decision variables must satisfy, such as resource availability, capacity limits, or regulatory requirements.

Having established optimization as a tool to formalize scheduling challenges, we now explore various **scheduling types and approaches**.

## 4 Scheduling Types and Approaches

### 4.1 Static vs. Dynamic Scheduling

#### 4.1.1 Static

Static scheduling refers to a scheduling approach where tasks or processes are assigned to resources (e.g., processors, machines) in advance, before the system begins execution. The schedule is predetermined and does not change during runtime. This method is often used in systems with predictable workloads and is efficient for scenarios where all task requirements and constraints are known beforehand [4].

### 4.1.2 Dynamic

Dynamic scheduling refers to a scheduling approach where tasks or processes are assigned to resources during runtime, based on the current state of the system and real-time conditions. This method is flexible and adapts to changes in workload, resource availability, or system priorities. It is commonly used in systems with unpredictable or varying workloads, such as real-time systems or cloud computing environments [4].

Moving from static and dynamic approaches, we now consider how **certainty** in task parameters influences scheduling decisions.

## 4.2 Deterministic vs. Stochastic Scheduling

### 4.2.1 Deterministic

Deterministic scheduling refers to a scheduling approach where all parameters of the problem, such as task durations, resource requirements, and constraints, are known with certainty in advance. This allows for the creation of a precise and fixed schedule before execution begins. It is commonly used in environments where tasks and resources are predictable, such as manufacturing or project management [3].

**Example:** In airline flight scheduling, fixed flight times, known crew assignments, and planned maintenance allow for precise scheduling.

### 4.2.2 Stochastic

Stochastic scheduling refers to a scheduling approach where some or all parameters of the problem, such as task durations, resource availability, or arrival times, are uncertain and modeled as random variables. This method incorporates probability distributions to account for variability and aims to optimize schedules under uncertainty. It is often used in systems with unpredictable elements, such as service industries, healthcare, or supply chain management [3].

**Example:** In emergency medical services, ambulances are dispatched based on real-time incoming requests, which are unpredictable in nature.

Having considered these contrasting scheduling approaches, we next examine the impact of **information availability** on scheduling decisions.

## 4.3 Offline vs. Online Scheduling

In scheduling theory, the distinction between offline and online scheduling lies in the availability of job information before decision-making:

### 4.3.1 Offline Scheduling

All job data (such as processing times, deadlines, and release dates) is known in advance. This allows for optimized, well-planned schedules. Offline scheduling is commonly used in manufacturing, logistics, and project management [5].

### 4.3.2 Online Scheduling

Jobs arrive dynamically, and decisions must be made without knowledge of future tasks. This is relevant in real-time systems, cloud computing, and dynamic environments like ride-sharing services. Algorithms for online scheduling often aim to minimize the worst-case performance [5].

With the temporal aspects of scheduling addressed, we now turn our focus to **the configuration** of resources.

## 4.4 Single-Machine vs. Multi-Machine Scheduling

### 4.4.1 Single Machine

All jobs are processed on a single machine, one at a time. The focus is on optimizing criteria such as minimizing completion time, lateness, or tardiness. This is the simplest scheduling environment and serves as a foundation for more complex models [1].

### 4.4.2 Multi-Machine

Jobs are processed on multiple machines, which can be arranged in different configurations, such as parallel machines, flow shops, job shops, or open shops. This type of scheduling introduces complexities like machine assignment, sequencing, and synchronization between tasks [1].

Continuing our discussion, we now detail two important **models** in multi-machine scheduling.

## 4.5 Flow Shop and Job Shop Scheduling

### 4.5.1 Flow Shop

Jobs follow the same machine sequence in a predefined order. Each job visits every machine in the same sequence, making this model common in assembly lines and manufacturing (e.g., car production). Optimization focuses on minimizing makespan, idle time, or flow time [1].

### 4.5.2 Job Shop

Jobs have unique machine sequences, meaning different jobs may follow different routes. This model applies to custom manufacturing and service industries (e.g., metal fabrication, hospitals). Solving job shop scheduling problems is computationally complex (often NP-hard), requiring heuristic or metaheuristic approaches [1].

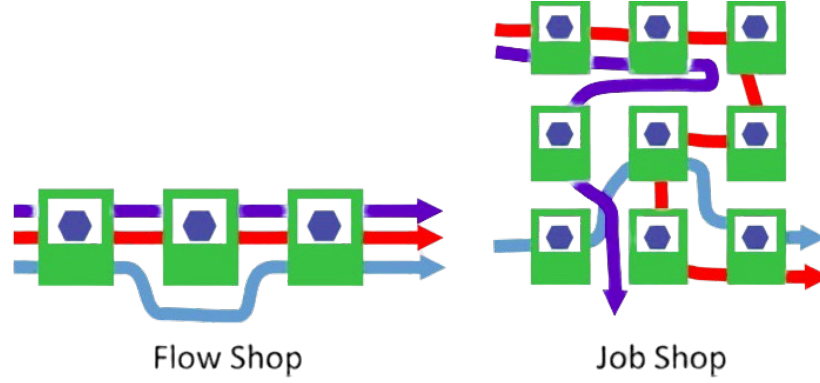


Figure 1: Flow Shop vs. Job shop [1]

## 5 Real-World Applications of Task Scheduling

Task scheduling theory finds extensive practical applications across numerous industries, where efficient allocation of limited resources to tasks directly impacts operational efficiency, cost reduction, and service quality. This section explores how scheduling principles are implemented in various real-world domains, demonstrating the practical significance of the concepts discussed in this chapter.

### 5.1 Manufacturing and Production Planning

In manufacturing environments, task scheduling plays a critical role in optimizing production processes and resource utilization [30]:

- **Production Line Scheduling:** Manufacturers use scheduling algorithms to determine the optimal sequence of jobs on assembly lines, minimizing setup times between different product types and reducing idle time of expensive machinery.
- **Just-in-Time Manufacturing:** Companies like Toyota implement sophisticated scheduling systems to ensure components arrive exactly when needed in the production process, reducing inventory costs while maintaining production flow.
- **Preventive Maintenance:** Scheduling regular maintenance activities during planned downtime minimizes disruption to production and extends equipment lifespan, balancing immediate production needs with long-term equipment reliability.



- **Industry 4.0 Integration:** Modern smart factories employ real-time scheduling systems that dynamically adjust production plans based on machine status, inventory levels, and incoming orders, creating adaptive manufacturing environments [31].

## 5.2 Healthcare Systems

Healthcare facilities face complex scheduling challenges involving patients, medical staff, equipment, and facilities [32]:

- **Patient Appointment Scheduling:** Hospitals and clinics use scheduling algorithms to optimize patient flow, reduce waiting times, and maximize the utilization of medical staff and equipment.
- **Operating Room Scheduling:** Surgical departments implement block scheduling or open scheduling approaches to allocate operating rooms, considering factors such as procedure duration, surgeon availability, and emergency cases.
- **Staff Rostering:** Healthcare facilities create work schedules for doctors, nurses, and support staff that satisfy coverage requirements, labor regulations, and staff preferences while ensuring patient care quality.
- **Emergency Response:** Ambulance services use dynamic scheduling algorithms to dispatch vehicles and personnel efficiently, considering factors like location, severity of emergencies, and available resources [33].

## 5.3 Cloud Computing and IT Services

In cloud computing environments, task scheduling algorithms optimize the allocation of computational resources [34]:

- **Virtual Machine Allocation:** Cloud service providers like AWS and Google Cloud use sophisticated scheduling algorithms to assign virtual machines to physical servers, balancing workload across data centers.
- **Job Scheduling in Distributed Systems:** Frameworks like Hadoop and Spark implement scheduling policies to distribute computational tasks across clusters of computers, considering data locality, resource availability, and job priorities.
- **Energy-Efficient Computing:** Data centers employ scheduling techniques that consolidate workloads on fewer servers during periods of low demand, allowing unused servers to be powered down to reduce energy consumption.

- **Quality of Service Management:** Cloud providers implement scheduling policies that ensure service level agreements (SLAs) are met by prioritizing critical tasks and allocating resources based on customer requirements [35].

## 5.4 Transportation and Logistics

The transportation and logistics sector relies heavily on scheduling to optimize the movement of goods and people [36]:

- **Vehicle Routing and Scheduling:** Delivery companies like FedEx and UPS use advanced algorithms to determine optimal routes and schedules for their fleets, considering factors such as delivery windows, traffic conditions, and vehicle capacity.
- **Airline Scheduling:** Airlines create complex schedules for aircraft, crew, and maintenance activities, balancing factors such as airport slots, crew duty time limitations, and aircraft utilization.
- **Public Transportation:** Transit authorities schedule buses, trains, and other public transportation services to meet passenger demand while minimizing operational costs and vehicle requirements.
- **Warehouse Operations:** Distribution centers schedule receiving, picking, packing, and shipping activities to optimize labor utilization and meet order fulfillment deadlines [37].

These real-world applications demonstrate how the theoretical concepts of task scheduling translate into practical solutions for complex resource allocation problems across diverse industries. As organizations continue to face increasing pressure to optimize operations and reduce costs, the importance of effective scheduling techniques will only grow, driving further innovation in both theory and practice.

Following our review of scheduling types and real-world applications, we now explore the **mathematical and algorithmic** foundations that underpin these approaches.

# 6 Mathematical and Algorithmic Foundations

## 6.1 Graph theory

Graph theory is a branch of discrete mathematics that studies structures called graphs, which consist of vertices (nodes) and edges (connections) between them. It is widely used in computer science, operations research, network analysis, and optimization.

**Directed vs. Undirected Graphs:** In directed graphs (digraphs), edges have a direction ( $A \rightarrow B$ ), while in undirected graphs, edges have no direction ( $A - B$ ).

**Weighted vs. Unweighted Graphs:** Weighted graphs assign values (costs, distances, etc.) to edges, whereas unweighted graphs do not.

**Paths and Cycles:** A path is a sequence of vertices connected by edges, while a cycle is a path that starts and ends at the same vertex.

**Trees and Connectivity:** A tree is a connected graph with no cycles, and a graph is connected if there is a path between any two vertices.

### Applications:

Shortest Path Algorithms: Dijkstra's and Bellman-Ford for finding optimal routes.

Network Flow Problems: Max-flow/min-cut applications in logistics and telecommunication.

Graph Coloring: Used in scheduling, register allocation, and map coloring.

Social Network Analysis: Studying relationships and influence between entities.

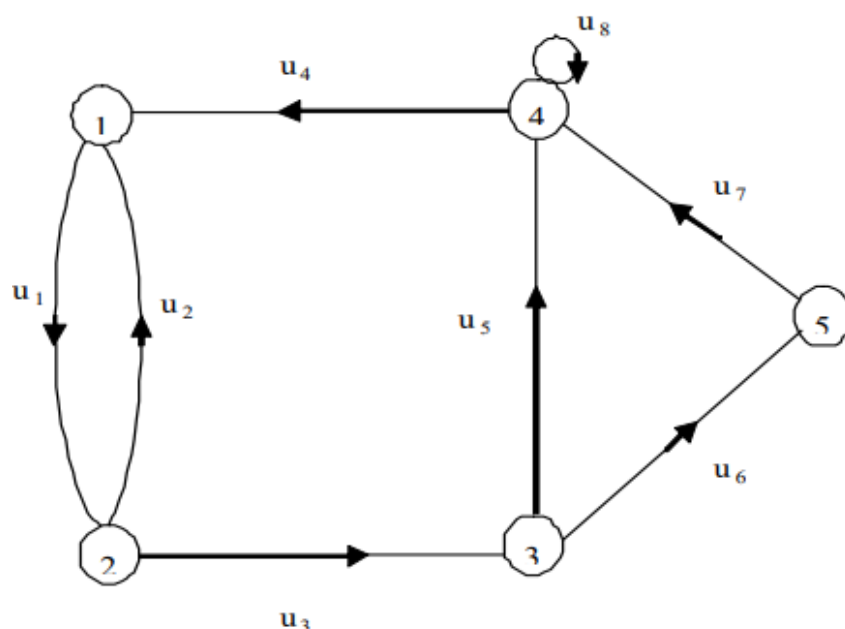


Figure 2: Graph example with directed edges [29]

## 6.2 Complexity Theory

Complexity theory is a branch of computer science and mathematics that studies the resources (time, space) needed to solve computational problems, classifying them by difficulty (e.g., P, NP, NP-complete). It determines if problems are tractable (efficiently solvable) or intractable [6].

### 6.2.1 P Problems

**P** (Polynomial time) problems are those that can be solved by a deterministic Turing machine in polynomial time. This means that there exists an algorithm that can solve

the problem in time proportional to a polynomial function of the input size. Examples include sorting a list of numbers or finding the shortest path in a graph.

### 6.2.2 NP Problems

**NP** (Nondeterministic Polynomial time) problems are those for which a proposed solution can be verified by a deterministic Turing machine in polynomial time. While it is not known whether these problems can be solved in polynomial time, they can be solved by a nondeterministic Turing machine in polynomial time. Examples include the Traveling Salesman Problem (TSP) and the Boolean Satisfiability Problem (SAT).

### 6.2.3 NP-Completeness

A problem is NP-complete if it is both in NP and NP-hard. NP-complete problems are the most difficult problems in NP, and they have the property that any problem in NP can be reduced to them in polynomial time. This means that if a polynomial-time algorithm is found for any NP-complete problem, then all NP problems can be solved in polynomial time. Examples of NP-complete problems include the Knapsack Problem, the Hamiltonian Cycle Problem, and the Graph Coloring Problem.

### 6.2.4 NP-Hard Problems

**NP-hard** problems are at least as hard as the hardest problems in NP, but they do not necessarily belong to NP themselves. This means that they may not have a polynomial-time verification algorithm. NP-hard problems include optimization problems like the Traveling Salesman Problem (TSP) and the Integer Linear Programming Problem (ILP).

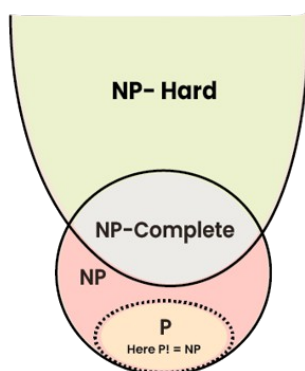


Figure 3: Complexity classes [6]

## 6.3 Heuristics and Metaheuristics

[9] Optimization problems, particularly in scheduling, often require efficient methods to find high-quality solutions within reasonable time constraints. Two major categories of approximation techniques are **heuristics** and **metaheuristics**, which aim to provide practical solutions when exact methods are computationally prohibitive.

### 6.3.1 Heuristics

Heuristics are problem-specific strategies that use predefined rules or shortcuts to construct feasible solutions quickly. They are designed for efficiency but do not guarantee optimality and may become trapped in local optima. Common heuristics in scheduling include:

- **Earliest Due Date (EDD)**: Prioritizes tasks with the earliest deadlines.
- **Shortest Processing Time (SPT)**: Prefers tasks with the shortest durations.
- **Longest Processing Time (LPT)**: Allocates longer tasks first to balance workload distribution.

While heuristics are computationally efficient, their effectiveness depends on the problem structure and the quality of their predefined rules.

### 6.3.2 Metaheuristics

Metaheuristics are high-level optimization strategies that guide heuristics to explore the solution space more effectively. Unlike heuristics, they incorporate mechanisms to escape local optima by balancing **exploration** (diversification) and **exploitation** (intensification). Common metaheuristic approaches include:

- **Genetic Algorithms (GA)**: Evolutionary algorithms inspired by natural selection.
- **Simulated Annealing (SA)**: A probabilistic method mimicking the annealing process in metallurgy.
- **Ant Colony Optimization (ACO)**: Uses artificial pheromones to guide search behavior.
- **Particle Swarm Optimization (PSO)**: Models collective intelligence inspired by swarm behavior.
- **Tabu Search (TS)**: Maintains a memory structure to avoid revisiting previous solutions.

Metaheuristics provide a more flexible and adaptive approach, making them suitable for complex scheduling problems where heuristic methods alone may struggle.

By understanding the fundamental differences between heuristics and metaheuristics, we can explore how these approaches are applied to scheduling problems in the following chapters.

# Chapter 2: Methodologies and Algorithms

This chapter explores key **methodologies**, starting with exact algorithms like Linear Programming and Branch and Bound, which ensure optimality but face scalability issues. We then cover heuristic and metaheuristic approaches, including List Scheduling, Genetic Algorithms, and Simulated Annealing, which offer efficient, near-optimal solutions. Finally, we examine real-world applications in manufacturing, cloud computing, and logistics, along with performance evaluation metrics to assess scheduling efficiency.

## 7 Difference Between Exact, Heuristic, and Metaheuristic Approaches

In combinatorial optimization and task scheduling problems, different solution approaches can be classified into three main categories: exact algorithms, heuristics, and metaheuristics. Each of these methods has distinct characteristics in terms of optimality, computational efficiency, and applicability.

**Exact Algorithms:** These methods guarantee an optimal solution by systematically exploring the entire solution space. Examples include dynamic programming, branch and bound, and integer linear programming (ILP). While highly accurate, their computational cost grows exponentially with problem size, making them impractical for large-scale instances.

**Heuristics:** Heuristic methods provide fast, practical solutions by following problem-specific rules or approximations. Unlike exact algorithms, they do not guarantee optimality but significantly reduce computational time. Examples include the nearest neighbor algorithm for the traveling salesman problem and the shortest processing time (SPT) rule in scheduling. Heuristics are efficient but lack generality, meaning they must be adapted to different problem structures.

**Metaheuristics:** These are high-level strategies designed to explore the solution space efficiently while avoiding local optima. Metaheuristics, such as genetic algorithms, simulated annealing, and particle swarm optimization, balance exploration and exploitation, making them applicable to a wide range of optimization problems. Although they often yield near-optimal solutions, they require careful parameter tuning and higher computational effort compared to simple heuristics.

Table 1 summarizes the key differences between these approaches:

Approach	Optimality	Speed	Applicability
Exact	Guaranteed	Slow for large problems	General
Heuristic	Not guaranteed	Fast	Problem-specific
Metaheuristic	Near-optimal	Moderate	General

Table 1: Comparison of Exact, Heuristic, and Metaheuristic Approaches

## 8 Exact algorithms

Exact algorithms guarantee finding an optimal solution and are suitable for small to moderate problem instances. This section discusses three prominent exact approaches: Linear and Integer Programming, Branch and Bound, and Dynamic Programming. These methods ensure optimality but may have high computational costs, making them applicable mainly for smaller or well-structured problems.

### 8.1 Linear Programming (LP) and Integer Programming (IP)

Linear Programming (LP) and Integer Programming (IP) are widely used mathematical optimization techniques for scheduling problems.

#### Linear Programming (LP)

LP is used when the scheduling problem can be formulated with linear constraints and a linear objective function. The general LP formulation is:

$$\text{Minimize } c^T x \tag{1}$$

subject to:

$$Ax \leq b, \quad x \geq 0 \tag{2}$$

where  $A$  is a constraint matrix,  $b$  is a constraint vector, and  $c$  is a cost coefficient vector. The Simplex method [13] and Interior Point methods [14] are commonly used to solve LP problems efficiently.

#### Integer Programming (IP)

Many scheduling problems require integer decision variables (e.g., assigning tasks to time slots). Integer Programming (IP) extends LP by restricting some or all variables to integer values:

$$\text{Minimize } c^T x \tag{3}$$



subject to:

$$Ax \leq b, \quad x \in Z^n \quad (4)$$

Mixed-Integer Programming (MIP) allows both integer and continuous variables and is often solved using Branch and Bound (B&B) [15].

Aspect	Description
LP in Scheduling	Useful for resource allocation, time optimization, and continuous variables
IP in Scheduling	Essential for task assignment, sequencing decisions, and discrete time slots
Computational Complexity	LP: Polynomial time; IP: NP-hard (exponential time in worst case)
Solution Methods	LP: Simplex, Interior Point; IP: Branch and Bound, Cutting Plane

Table 2: Comparison of LP and IP in Scheduling Problems

## 8.2 Branch and Bound (B&B)

Branch and Bound is an exact method for solving combinatorial optimization problems, including scheduling [16]. It systematically explores the solution space by:

1. Branching: Dividing the problem into smaller subproblems.
2. Bounding: Computing bounds on the optimal solution in each subproblem.
3. Pruning: Eliminating subproblems that cannot lead to a better solution than the best-known one.

B&B is often combined with relaxation techniques, such as LP relaxation, to improve efficiency [17]. It is particularly useful for solving Integer Programming problems efficiently.

Step	Description
Initialization	Start with the full problem as the root node.
Branching	Divide the problem into smaller subproblems.
Bounding	Calculate bounds for each subproblem.
Pruning	Discard subproblems that exceed current best solution.
Optimality Check	If no subproblems remain, return the best solution found.

Table 3: Branch and Bound Strategy

### 8.3 Dynamic Programming (DP)

Dynamic Programming (DP) solves problems by breaking them into smaller subproblems and solving each only once [18]. It is particularly useful for scheduling problems with optimal substructure and overlapping subproblems. The DP approach follows these steps:

1. Define the state variables representing the problem at different stages.
2. Establish the recurrence relation between states.
3. Use memoization or bottom-up computation to avoid redundant calculations.

A well-known example is the Weighted Interval Scheduling problem, solved efficiently using DP [19]. DP is highly effective when a problem can be broken into overlapping subproblems, significantly reducing computational complexity compared to brute-force approaches.

**Exact algorithms** such as LP/IP, B&B, and DP provide optimal solutions for scheduling problems but may be computationally expensive for large instances. Hybrid approaches combining these methods with heuristics are often used in practice to balance accuracy and efficiency.

## 9 Heuristic Approaches

Heuristic approaches provide fast and practical solutions to scheduling problems by following predefined rules and approximations. While they do not guarantee optimality, they offer efficient solutions for large-scale problems where exact methods are computationally expensive.

### 9.1 List Scheduling

List Scheduling is a simple and widely used heuristic approach for task scheduling. It operates by maintaining a priority list of tasks and assigning them to available resources based on predefined rules. Common list scheduling strategies include:

- **Earliest Due Date (EDD):** Tasks are scheduled in ascending order of their due dates. This method minimizes the maximum lateness in scheduling problems [20].
- **Shortest Processing Time (SPT):** Tasks with the shortest processing times are scheduled first. This approach minimizes the average completion time [21].

- **Longest Processing Time (LPT):** Tasks with the longest processing times are scheduled first. It is often used to balance workload distribution across machines [22].

The List Scheduling approach is computationally efficient and can be implemented in  $O(n \log n)$  time complexity when using a priority queue.

## 9.2 Greedy Algorithms

Greedy algorithms are a class of heuristic methods that make locally optimal choices at each step with the hope of finding a global optimum. They are widely used in scheduling problems, such as task scheduling, job sequencing, and resource allocation. The key idea behind greedy algorithms is to make the best possible decision at each step without considering the overall problem, which often leads to efficient and practical solutions.

### Key Characteristics of Greedy Algorithms:

- **Local Optimality:** At each step, the algorithm selects the best available option based on a predefined criterion (e.g., shortest processing time, earliest deadline, etc.).
- **No Backtracking:** Once a decision is made, it is not revisited, which makes greedy algorithms computationally efficient.
- **Applicability:** Greedy algorithms are particularly useful for problems where a globally optimal solution can be constructed from a series of locally optimal choices.

### Examples of Greedy Algorithms in Scheduling:

- **Shortest Job First (SJF):** A scheduling algorithm that selects the task with the shortest processing time first. This approach minimizes the average waiting time and is commonly used in operating systems and job scheduling.
- **Knapsack Problem:** A greedy approach can be used to solve the fractional knapsack problem, where items are selected based on their value-to-weight ratio.
- **Interval Scheduling:** In interval scheduling problems, greedy algorithms can be used to select the maximum number of non-overlapping tasks based on their finish times.

### Limitations of Greedy Algorithms:

- Greedy algorithms do not always guarantee an optimal solution, especially for problems where local optimal choices do not lead to a global optimum.

- They are problem-specific and may require careful design of the selection criterion to achieve good results.

Greedy algorithms are widely used in real-world applications, such as task scheduling in operating systems, job sequencing in manufacturing, and resource allocation in cloud computing. Their simplicity and efficiency make them a popular choice for solving complex scheduling problems.

### 9.3 Priority-based Scheduling

Priority-based scheduling assigns tasks based on a predefined priority scheme. Priorities can be static (fixed at the start) or dynamic (adjusted during execution). Some common priority-based strategies include:

- **Weighted Priority Scheduling:** Each task is assigned a weight based on factors such as importance, deadline, or resource requirements. Tasks with higher weights are scheduled first [23].
- **Critical Path Method (CPM):** Tasks on the longest path in a directed acyclic graph (DAG) representation of the scheduling problem are given the highest priority to avoid delays [24].
- **Earliest Start Time (EST):** Tasks are scheduled in the order of their earliest possible start time, considering resource constraints [1].
- **Program Evaluation and Review Technique (PERT):** PERT is a project management tool used to schedule, organize, and coordinate tasks within a project. It is particularly useful for managing complex projects with uncertain task durations. PERT uses a probabilistic approach to estimate task durations, considering three time estimates for each task:
  - **Optimistic Time (O):** The minimum possible time required to complete a task.
  - **Most Likely Time (M):** The best estimate of the time required to complete a task.
  - **Pessimistic Time (P):** The maximum possible time required to complete a task.

The expected time (TE) for each task is calculated using the formula:

$$TE = \frac{O + 4M + P}{6}$$

PERT also identifies the **critical path**, which is the longest sequence of tasks that determines the minimum project duration. Tasks on the critical path are given the highest priority, as any delay in these tasks will directly impact the project's completion time. PERT is widely used in industries such as construction, software development, and manufacturing, where project timelines are critical and task durations are uncertain.

Priority-based scheduling is commonly used in real-time systems, cloud computing, and industrial production planning due to its adaptability and efficiency.

## 10 Metaheuristic Approaches

Metaheuristic approaches are high-level strategies designed to explore the solution space efficiently while avoiding local optima. These methods balance exploration and exploitation, making them suitable for a wide range of optimization problems. Below, we discuss some of the most widely used metaheuristic techniques for scheduling problems.

### 10.1 Genetic Algorithms (GA)

Genetic Algorithms (GA) are inspired by the principles of natural evolution. They operate by maintaining a population of potential solutions, evolving them through selection, crossover, and mutation [25].

#### Key Steps:

- **Initialization:** Generate an initial population of solutions.
- **Selection:** Choose the best individuals based on a fitness function.
- **Crossover:** Combine features from selected individuals to create offspring.
- **Mutation:** Introduce small random changes to maintain diversity.
- **Termination:** Stop when a convergence criterion is met.

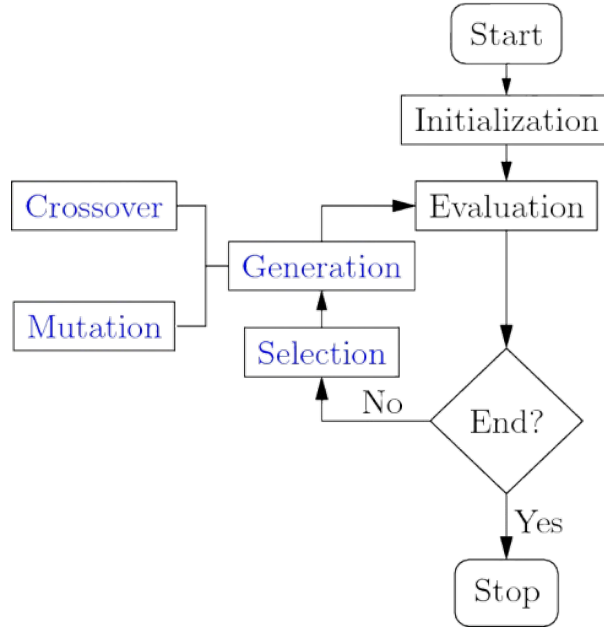


Figure 4: The basic process of genetic algorithm [7]

GA is widely used in scheduling due to its ability to find near-optimal solutions efficiently [7].

## 10.2 Simulated Annealing (SA)

Simulated Annealing (SA) is a probabilistic optimization technique inspired by the annealing process in metallurgy [8]. It allows occasional acceptance of worse solutions to escape local optima.

### Key Concepts:

- **Temperature:** Controls the probability of accepting worse solutions.
- **Cooling Schedule:** Gradually reduces the temperature over iterations.
- **Acceptance Probability:** Determines whether a worse solution is accepted based on the Metropolis criterion.

SA is particularly effective for complex scheduling problems where the solution space is large and contains many local optima.

## 10.3 Ant Colony Optimization (ACO)

Ant Colony Optimization (ACO) is inspired by the foraging behavior of ants [26]. It uses artificial pheromones to guide the search towards promising regions of the solution space.

### Key Steps:

- **Initialization:** Place ants randomly in the search space.

- **Pheromone Update:** Reinforce paths that lead to good solutions.
- **Exploration and Exploitation:** Balance between exploring new paths and following high-pheromone trails.
- **Convergence:** Stop when the best solution stabilizes.

ACO is highly effective for scheduling problems that can be modeled as graph traversal tasks.

## 10.4 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is inspired by the collective movement of birds and fish [27]. It optimizes a population of solutions (particles) by updating their positions based on personal and global best values.

### Key Components:

- **Particles:** Represent potential solutions.
- **Velocity Update:** Adjusts each particle's movement based on past experiences and the best global solution.
- **Convergence:** Occurs when all particles cluster around an optimal solution.

PSO is useful for scheduling problems requiring continuous optimization.

## 10.5 Tabu Search (TS)

Tabu Search (TS) is an iterative local search algorithm that uses a memory structure to avoid cycling back to previously visited solutions [28].

### Key Features:

- **Tabu List:** Stores recently visited solutions to prevent revisits.
- **Aspiration Criteria:** Allows overriding tabu status if a move leads to a better solution.
- **Diversification and Intensification:** Ensures exploration of new areas while refining good solutions.

TS is widely used in scheduling due to its ability to escape local optima efficiently.

## 11 Performance and Evaluation Metrics

### 11.1 Makespan

**Makespan** is the total time required to complete all scheduled jobs in a system. It represents the time from the start of the first task to the completion of the last one, making it a key performance metric in **scheduling and optimization** [1].

#### Mathematical Definition

The makespan  $C_{\max}$  is given by:

$$C_{\max} = \max(C_1, C_2, \dots, C_n) \quad (5)$$

where  $C_i$  is the completion time of job  $i$ .

### 11.2 Throughput

**Throughput** is a performance metric that measures the rate at which a system produces goods or completes tasks. It is commonly defined as the number of items processed per unit of time [1].

#### Mathematical Representation

If  $N$  is the number of completed items over a time period  $T$ , then throughput  $\theta$  is given by:

$$\theta = \frac{N}{T}$$

where:

- $N$  is the total number of items processed,
- $T$  is the total time taken.

### 11.3 Response Time

**Response Time** is the duration between the initiation of a request and the delivery of the corresponding response. It is an essential metric for evaluating the performance of systems in computing, networking, and service operations [10].



## Mathematical Representation

If  $T_{\text{start}}$  denotes the time when a request is initiated and  $T_{\text{finish}}$  is the time when the response is received, the response time  $R$  is given by:

$$R = T_{\text{finish}} - T_{\text{start}}$$

### 11.4 Latency

**Latency** refers to the time delay between the initiation of a process and its completion. It is a critical performance metric in computing, networking, and manufacturing [10].

## Mathematical Representation

Latency  $L$  can be expressed as:

$$L = T_{\text{response}} - T_{\text{request}}$$

where:

- $T_{\text{request}}$  is the time when a request is made,
- $T_{\text{response}}$  is the time when the response is received.

### 11.5 Load Balancing

**Load Balancing** is the process of distributing work evenly across multiple resources (such as servers, machines, or processors) to achieve optimal resource utilization, maximize throughput, minimize response time, and prevent any single resource from becoming a bottleneck [11].

## Mathematical Representation

If  $L_i$  represents the load on resource  $i$  for  $i = 1, 2, \dots, n$ , then a balanced system aims to satisfy:

$$L_1 \approx L_2 \approx \dots \approx L_n \approx \frac{L_{\text{total}}}{n}$$

where  $L_{\text{total}}$  is the total load distributed across all resources.

### 11.6 Fairness

**Fairness** in scheduling and resource allocation refers to the equitable distribution of resources among competing tasks or users, ensuring that no single entity is unduly favored [12].

## Conceptual Definition

Fairness aims to balance the allocation of resources (e.g., CPU time, bandwidth) so that all tasks or users receive a reasonable share, minimizing the risk of starvation.

## Mathematical Representation

A common metric for quantifying fairness is **Jain's Fairness Index**, defined as:

$$J(x_1, x_2, \dots, x_n) = \frac{(\sum_{i=1}^n x_i)^2}{n \sum_{i=1}^n x_i^2}$$

where:

- $x_i$  is the resource allocation for the  $i$ -th task or user,
- $n$  is the total number of tasks or users.

The index ranges from  $\frac{1}{n}$  (worst case) to 1 (best case, perfect fairness).

# Chapter 3: Results and Discussion

This chapter presents the implementation, results, and analysis of an exact approach to solving the Job Shop Scheduling Problem (JSSP) using Integer Linear Programming. We begin with a detailed explanation of our problem instance, followed by a comprehensive description of the implementation. We then analyze the results, highlighting the strengths and limitations of the approach.

## 12 Problem Definition

For our experimental analysis, we designed a moderately complex Job Shop Scheduling Problem instance that is sufficiently challenging to demonstrate the capabilities of our algorithm while remaining tractable for the exact method. The problem consists of scheduling a set of jobs, each comprising a sequence of operations that must be processed on specific machines in a predetermined order.

### 12.1 Problem Parameters

Our test instance consists of 5 jobs and 3 machines, with the following parameters:

- $n = 5$ : Number of jobs
- $m = 3$ : Number of machines
- $p_{ij}$ : Processing time of operation  $j$  of job  $i$
- $M_{ij}$ : Machine required for operation  $j$  of job  $i$

### 12.2 Constraints

The problem is subject to the following constraints:

1. Each operation must be processed without interruption (non-preemptive scheduling)
2. Each machine can process only one operation at a time
3. Operations of the same job must be processed in the given order (precedence constraints)
4. All jobs are available at time zero (no release dates)
5. No machine can process more than one operation simultaneously

### 12.3 Objective Function

The primary objective is to minimize the makespan, which is the completion time of the last operation to finish:

$$\text{Minimize } C_{\max} = \max_{i \in \{1, 2, \dots, n\}} C_i \quad (6)$$

where  $C_i$  is the completion time of job  $i$ .

## 13 Exact Approach: Integer Linear Programming

Integer Linear Programming (ILP) is an exact optimization method that guarantees finding the optimal solution, provided sufficient computational resources. For the JSSP, we formulated the problem as a mixed-integer linear program with binary decision variables.

### 13.1 Mathematical Formulation

Our ILP formulation for the JSSP is as follows:

#### 13.1.1 Decision Variables

- $S_{ij}$ : Start time of operation  $j$  of job  $i$
- $x_{i_1 j_1 i_2 j_2}$ : Binary variable that equals 1 if operation  $j_1$  of job  $i_1$  precedes operation  $j_2$  of job  $i_2$  on the same machine, and 0 otherwise
- $C_{\max}$ : Makespan (completion time of the last operation)

#### 13.1.2 Objective Function

$$\text{Minimize } C_{\max} \quad (7)$$

#### 13.1.3 Constraints

1. Precedence constraints within jobs:

$$S_{i,j+1} \geq S_{ij} + p_{ij} \quad \forall i \in \{1, 2, \dots, n\}, j \in \{1, 2, \dots, m-1\} \quad (8)$$

2. Makespan constraint:

$$C_{\max} \geq S_{i,m} + p_{i,m} \quad \forall i \in \{1, 2, \dots, n\} \quad (9)$$

3. Non-overlapping constraints for operations on the same machine:

$$S_{i_1j_1} + p_{i_1j_1} \leq S_{i_2j_2} + M(1 - x_{i_1j_1i_2j_2}) \quad (10)$$

$$S_{i_2j_2} + p_{i_2j_2} \leq S_{i_1j_1} + Mx_{i_1j_1i_2j_2} \quad (11)$$

for all operations  $(i_1, j_1)$  and  $(i_2, j_2)$  that use the same machine, where  $M$  is a large constant (big-M).

## 13.2 Python Implementation

We implemented the ILP formulation using the PuLP library, which provides an interface to various linear programming solvers. The following code snippet shows the core components of our implementation:

```

1  import pulp
2  import time
3
4  class JobShopSchedulingILP:
5      def __init__(self, n_jobs, n_machines, processing_times,
6          ↪ machine_order, big_m=10000):
7
8          Args:
9              n_jobs: Number of jobs
10             n_machines: Number of machines
11             processing_times: List of lists where
12             ↪ processing_times[i][j] is the processing time of
13             ↪ operation j of job i
14             machine_order: List of lists where machine_order[i][j] is
15             ↪ the machine required for operation j of job i
16             big_m: A large constant for the big-M method in ILP
17             ↪ constraints
18
19             """
20             # Initialize parameters
21             self.n_jobs = n_jobs
22             self.n_machines = n_machines
23             self.processing_times = processing_times
24             self.machine_order = machine_order
25             self.big_m = big_m

```

```

22     def solve(self, time_limit=300):
23         """Solve the JSSP using Integer Linear Programming."""
24         start_time_solve = time.time()
25
26         # Create the ILP problem
27         prob = pulp.LpProblem("JobShopScheduling", pulp.LpMinimize)
28
29         # Decision variables
30         # start_time[i][j] = start time of operation j of job i
31         start_time = {}
32         for i in range(self.n_jobs):
33             for j in range(self.n_machines):
34                 start_time[(i, j)] =
35                     ↪ pulp.LpVariable(f"start_time_{i}_{j}",
36                                     lowBound=0,
37                                     ↪ cat='Integer')
38
39         # Binary variables for sequencing operations on machines
40         x = {}
41         for i1 in range(self.n_jobs):
42             for j1 in range(self.n_machines):
43                 m1 = self.machine_order[i1][j1]
44                 for i2 in range(self.n_jobs):
45                     for j2 in range(self.n_machines):
46                         m2 = self.machine_order[i2][j2]
47                         # Only create variables for operations on the
48                         ↪ same machine
49                         if m1 == m2 and (i1 != i2 or j1 != j2):
50                             x[(i1, j1, i2, j2)] = pulp.LpVariable(
51                                 f"x_{i1}_{j1}_{i2}_{j2}", cat='Binary')
52
53         # Makespan variable
54         makespan = pulp.LpVariable("makespan", lowBound=0, cat='Integer')
55
56         # Objective: Minimize makespan
57         prob += makespan, "Minimize makespan"

```

```

58     # 1. Precedence constraints within jobs
59     for i in range(self.n_jobs):
60         for j in range(self.n_machines - 1):
61             prob += start_time[(i, j+1)] >= start_time[(i, j)] +
                ↪ self.processing_times[i][j]
62
63     # 2. Makespan constraint
64     for i in range(self.n_jobs):
65         prob += makespan >= start_time[(i, self.n_machines-1)] + \
66             self.processing_times[i][self.n_machines-1]
67
68     # 3. Non-overlapping constraints for operations on the same
        ↪ machine
69     for i1 in range(self.n_jobs):
70         for j1 in range(self.n_machines):
71             m1 = self.machine_order[i1][j1]
72             for i2 in range(self.n_jobs):
73                 for j2 in range(self.n_machines):
74                     m2 = self.machine_order[i2][j2]
75                     # Only add constraints for operations on the
                        ↪ same machine
76                     if m1 == m2 and (i1 != i2 or j1 != j2):
77                         # Ensure no overlap between operations on
                            ↪ the same machine
78                         prob += start_time[(i1, j1)] +
                            ↪ self.processing_times[i1][j1] <= \
79                             start_time[(i2, j2)] + self.big_m * (1
                                ↪ - x[(i1, j1, i2, j2)])
80
81                         prob += start_time[(i2, j2)] +
                            ↪ self.processing_times[i2][j2] <= \
82                             start_time[(i1, j1)] + self.big_m *
                                ↪ x[(i1, j1, i2, j2)]
83
84     # Solve the problem
85     solver = pulp.PULP_CBC_CMD(timeLimit=time_limit)
86     prob.solve(solver)
87
88     # Get solution time

```

```

89         solution_time = time.time() - start_time_solve
90
91         # Check if the problem was solved successfully
92         if pulp.LpStatus[prob.status] == 'Optimal':
93             # Extract the solution
94             schedule = {}
95             for i in range(self.n_jobs):
96                 for j in range(self.n_machines):
97                     schedule[(i, j)] = {
98                         'start_time': pulp.value(start_time[(i, j)]),
99                         'machine': self.machine_order[i][j],
100                        'processing_time': self.processing_times[i][j],
101                        'end_time': pulp.value(start_time[(i, j)]) +
102                        ↪ self.processing_times[i][j]
103                    }
104
105             # Return results
106             return {
107                 'status': pulp.LpStatus[prob.status],
108                 'makespan': pulp.value(makespan),
109                 'schedule': schedule,
110                 'solution_time': solution_time
111             }
112         else:
113             return {
114                 'status': pulp.LpStatus[prob.status],
115                 'solution_time': solution_time
116             }
117
118         # Example usage with the given parameters
119         def run_example():
120             # Parameters for 5 jobs and 3 machines example
121             n_jobs = 5
122             n_machines = 3
123
124             # Example processing times for each operation of each job
125             processing_times = [
126                 [10, 8, 4],    # Job 0

```



```

127         [4, 7, 9],      # Job 2
128         [12, 6, 5],     # Job 3
129         [7, 10, 4]      # Job 4
130     ]
131
132     # Example machine order for each operation of each job
133     machine_order = [
134         [0, 1, 2],       # Job 0
135         [0, 2, 1],       # Job 1
136         [1, 0, 2],       # Job 2
137         [1, 2, 0],       # Job 3
138         [2, 0, 1]        # Job 4
139     ]
140
141     # Create and solve the problem
142     jssp = JobShopSchedulingILP(n_jobs, n_machines, processing_times,
143                               ↪ machine_order)
144     result = jssp.solve()
145
146     # Print results
147     print(f"Status: {result['status']}")
148     print(f"Makespan: {result['makespan']} time units")
149     print(f"Solution time: {result['solution_time']:.2f} seconds")
150
151     return result
152
153 if __name__ == "__main__":
154     run_example()

```

### 13.3 Results and Analysis

We applied our ILP approach to the problem instance with 5 jobs and 3 machines. The algorithm found the optimal solution with a makespan of 49 time units in 0.11 seconds. Figure 5 shows the Gantt chart of the optimal schedule.

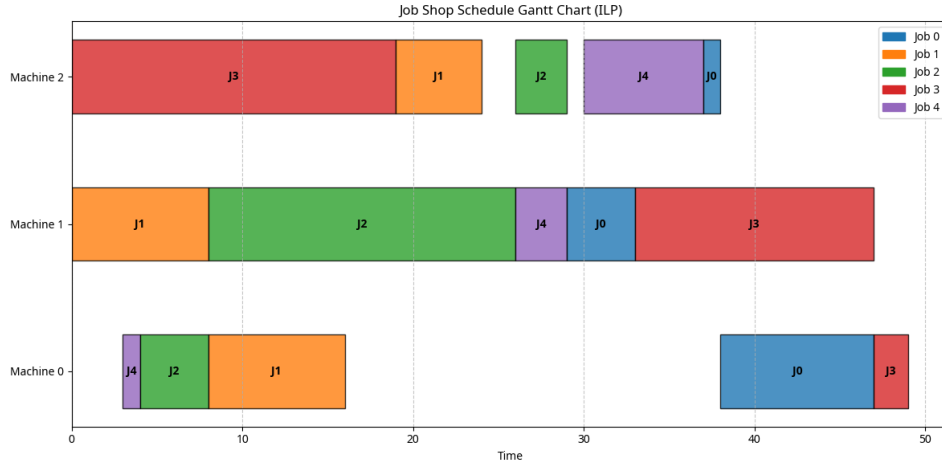


Figure 5: Gantt chart of the optimal schedule found by Integer Linear Programming

The ILP approach demonstrated several key characteristics:

- **Optimality:** The algorithm guarantees finding the optimal solution if given sufficient time.
- **Precision:** The solution satisfies all constraints exactly.
- **Determinism:** The algorithm produces the same result in repeated runs.

However, the ILP approach faces significant scalability challenges. The number of binary variables and constraints grows quadratically with the number of jobs and machines, making it impractical for large problem instances without decomposition or relaxation techniques.

## 14 Analysis of the Exact Approach

### 14.1 Strengths and Limitations of Integer Linear Programming

**Strengths:**

- Guarantees optimal solutions (given sufficient time)
- Provides optimality gaps for early termination
- Deterministic results
- Well-established mathematical foundation
- Precise handling of complex constraints
- Ability to prove optimality of solutions

**Limitations:**

- Limited scalability to large problem instances
- Exponential growth in computational requirements
- Difficulty in modeling certain complex constraints
- Sensitivity to the big-M parameter values
- Memory-intensive for large problem instances

This chapter has presented a comprehensive analysis of an exact approach to solving the Job Shop Scheduling Problem using Integer Linear Programming. Our implementation and experimental results demonstrate the strengths and limitations of this approach.

The ILP method provides optimal solutions with mathematical guarantees but faces scalability challenges for larger problem instances. The choice of this approach depends on the specific requirements of the application domain, including problem size, time constraints, and the importance of finding provably optimal solutions.

For practical applications with larger problem instances, decomposition techniques, problem-specific constraints, or relaxation methods can be employed to enhance the scalability of exact approaches. These techniques maintain the mathematical rigor of exact methods while addressing their computational limitations.

Future work could explore advanced decomposition techniques, column generation approaches, and cutting plane methods to enhance the performance of ILP on larger and more complex scheduling problems.

# Chapter 4: Task Scheduling Application

This chapter presents the **most significant contribution** of our project: a comprehensive task scheduling application that implements both exact and metaheuristic methods to solve complex scheduling problems. By combining the mathematical precision of Integer Linear Programming through PuLP with the adaptive exploration capabilities of Simulated Annealing, our application provides powerful and flexible scheduling solutions that minimize makespan while respecting task dependencies and resource constraints.

## 15 Introduction to the Application

The task scheduling application we have developed represents the practical culmination of the theoretical concepts and methodologies discussed in previous chapters. This application addresses the critical need for efficient task scheduling in project management, manufacturing, and software development environments where optimal resource utilization and minimized completion times are essential for operational success.

The primary objective of our application is to provide an intuitive and powerful tool that enables users to:

- Define tasks with specific durations, deadlines, and dependencies
- Manage various types of resources (personnel, machines, servers)
- Generate optimized schedules that minimize the overall project makespan
- Visualize scheduling results through interactive Gantt charts and timelines
- Compare different scheduling approaches (exact vs. metaheuristic)
- Export scheduling data for further analysis and integration

What distinguishes our application from existing scheduling tools is its dual-method approach to solving scheduling problems. By implementing both PuLP (an exact method) and Simulated Annealing (a metaheuristic method), the application offers users the flexibility to choose between mathematical optimality and computational efficiency based on their specific requirements and problem complexity.

## 16 Theoretical Foundation

The application is built upon the solid theoretical foundation established in previous chapters, particularly drawing from concepts in operations research, optimization theory, and scheduling algorithms. Before delving into the implementation details, it is important to briefly recap the key theoretical elements that underpin our application.

## 17 Application Architecture and Design

Our task scheduling application follows a modular architecture that separates concerns and promotes maintainability. The application is structured around several key components:

### 17.1 Core Components

- **Task Manager:** Handles the creation, modification, and deletion of tasks, including their properties (duration, deadline, dependencies)
- **Resource Manager:** Manages different types of resources (personnel, machines, servers) and their availability
- **Scheduler:** Implements both the PuLP and Simulated Annealing algorithms for generating optimized schedules
- **Visualization Engine:** Renders the scheduling results as interactive Gantt charts and timelines
- **Data Export Module:** Facilitates exporting scheduling data to external formats (e.g., Excel)

### 17.2 User Interface Design

The user interface is designed with simplicity and usability in mind, featuring:

- A clean, intuitive layout with clear navigation between different sections
- Visual indicators and icons that enhance understanding of task relationships and statuses
- Interactive elements that allow users to easily add, modify, and organize tasks and resources
- Responsive visualizations that provide immediate feedback on scheduling decisions

### 17.3 Data Model

The application's data model is centered around the following key entities:

- **Task:** Represents a unit of work with attributes such as name, description, duration, deadline, priority, and dependencies

- **Resource:** Represents a person, machine, or server with attributes such as name, type, cost, and availability
- **Schedule:** Represents a solution to the scheduling problem, containing task start times, resource assignments, and performance metrics

These entities and their relationships form the foundation of the application’s functionality, enabling the complex scheduling operations that the application performs.

## 18 Implementation of the Exact Method (PuLP)

The exact method implementation leverages the PuLP library to formulate and solve the task scheduling problem as an integer linear program. This approach guarantees finding the optimal solution that minimizes the makespan while respecting all constraints.

It provides optimal solutions for small to medium-sized problems but may become computationally expensive for larger instances due to the NP-hard nature of the scheduling problem.

## 19 Implementation of the Metaheuristic Method (Simulated Annealing)

For larger scheduling problems where exact methods become impractical, our application implements Simulated Annealing (SA), a powerful metaheuristic that can find near-optimal solutions in reasonable computation time.

This implementation allows the algorithm to escape local optima and explore the solution space more effectively, often leading to high-quality solutions even for complex scheduling problems.

## 20 User Interface and Experience

The application features a clean, intuitive user interface designed to simplify the complex task of schedule creation and optimization. The interface is organized into several key sections that guide users through the scheduling process.

### 20.1 Task Management Interface

The task management interface allows users to create, edit, and organize tasks with the following features:

- **Task Creation:** Users can add new tasks by specifying essential information such as name, description, duration, priority, and deadline.
- **Dependency Management:** The interface provides a straightforward way to establish dependencies between tasks, ensuring that precedence relationships are properly captured.
- **Visual Indicators:** Tasks are displayed with clear visual cues that indicate their status, priority, and relationships with other tasks.

As shown in Figure 6, the task creation dialog provides fields for all necessary task attributes and a checklist for selecting dependencies.

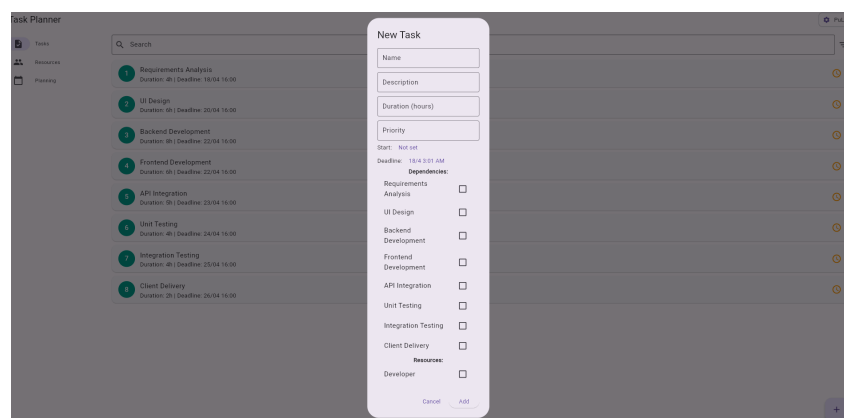


Figure 6: Task creation interface with fields for name, description, duration, priority, deadline, and dependencies

## 21 Method Selection in the Application

A distinctive feature of our task scheduling application is the ability for users to choose between two powerful scheduling methods: the exact method using PuLP and the metaheuristic approach using Simulated Annealing. This flexibility allows users to select the most appropriate method based on their specific scheduling requirements, problem complexity, and time constraints.

### 21.1 Choosing Between Exact and Metaheuristic Methods

The application provides a straightforward interface for method selection through a drop-down menu in the upper right corner of the application, as shown in Figure 7. This intuitive design allows users to easily switch between methods without navigating through complex menus or settings.

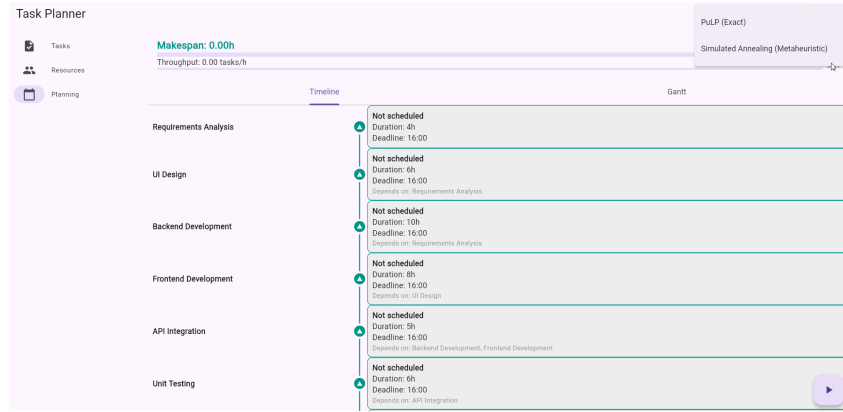


Figure 7: Method selection dropdown showing options for PuLP (Exact) and Simulated Annealing (Metaheuristic)

When users click on the method selector, they are presented with two options:

- **PuLP (Exact):** This option utilizes the PuLP library to formulate and solve the scheduling problem as an integer linear program, guaranteeing an optimal solution.
- **Simulated Annealing (Metaheuristic):** This option employs the Simulated Annealing algorithm to find near-optimal solutions efficiently, particularly valuable for larger scheduling problems.

## 21.2 Resource Management Interface

The resource management interface enables users to define and manage the resources available for task execution:

- **Resource Types:** The application supports different types of resources, including personnel, machines, and servers, each with specific attributes.
- **Cost Specification:** Users can assign cost values to resources, allowing for cost-aware scheduling decisions.
- **Resource Visualization:** Resources are displayed with distinctive icons that indicate their type, making it easy to identify different resource categories.

Figure 8 illustrates the resource creation dialog, which allows users to specify the resource name, type, and cost.

The resource management view (Figure 9) provides an overview of all available resources, organized by type with associated costs.



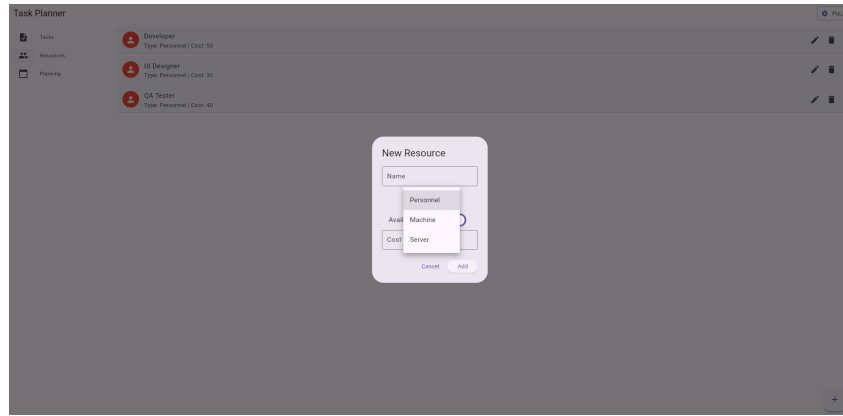


Figure 8: Resource creation interface with fields for name, type, and cost

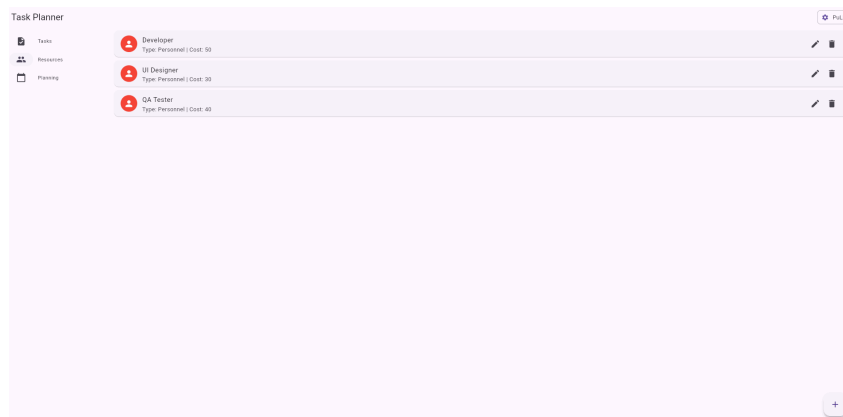


Figure 9: Resource management view showing different types of resources with their costs

### 21.3 Schedule Visualization

Once tasks and resources are defined, the application offers powerful visualization tools to represent the generated schedules:

- **Timeline View:** Shows tasks arranged chronologically with their dependencies and durations clearly indicated.
- **Gantt Chart:** Provides a traditional Gantt chart visualization that shows task durations, start times, and relationships.
- **Performance Metrics:** Displays key metrics such as makespan and throughput prominently at the top of the visualization.

Figure 10 shows the timeline view of an unscheduled project, with tasks listed along with their durations, deadlines, and dependencies.

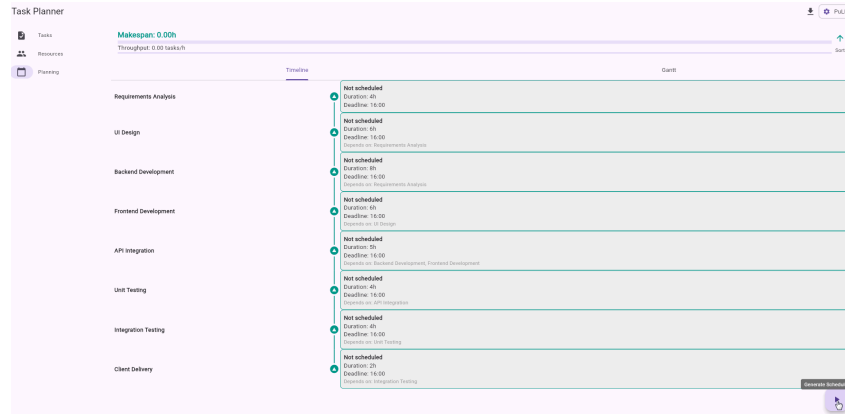


Figure 10: Timeline view before scheduling, showing tasks with their properties and dependencies

After scheduling, the application presents the optimized schedule with calculated start times for each task, as shown in Figure 11.

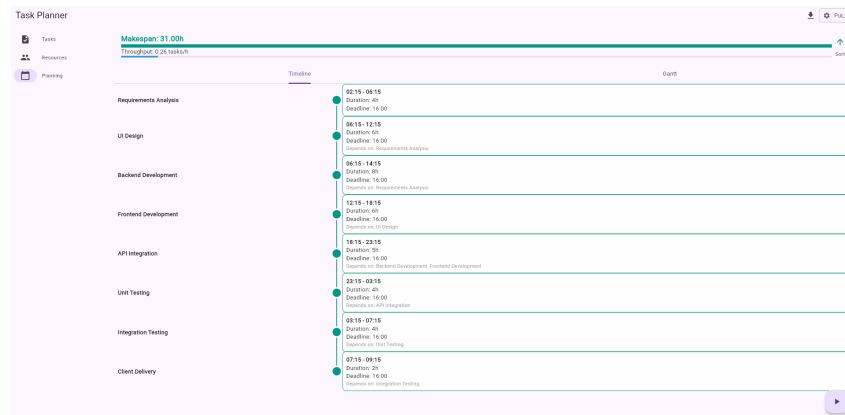


Figure 11: Timeline view after scheduling, showing the optimized task sequence with calculated start times

## 22 Application Features

Our task scheduling application offers a comprehensive set of features designed to address the complex requirements of modern scheduling problems.

### 22.1 Task Management

The application provides robust task management capabilities:

- **Flexible Task Definition:** Users can define tasks with various attributes, including name, description, duration, priority, and deadline.

- **Dependency Handling:** The application supports complex task dependencies, ensuring that precedence relationships are respected in the generated schedules.
- **Task Categorization:** Tasks can be categorized and filtered based on different criteria, making it easier to manage large projects.

## 22.2 Resource Management

Effective resource management is a key feature of the application:

- **Multiple Resource Types:** The application supports different types of resources (personnel, machines, servers), each with specific attributes and constraints.
- **Resource Allocation:** The scheduling algorithms intelligently allocate resources to tasks, ensuring optimal utilization while respecting capacity constraints.
- **Cost-Aware Scheduling:** By incorporating resource costs, the application can generate schedules that balance time and cost considerations.

## 22.3 Schedule Generation

The core functionality of the application is its ability to generate optimized schedules:

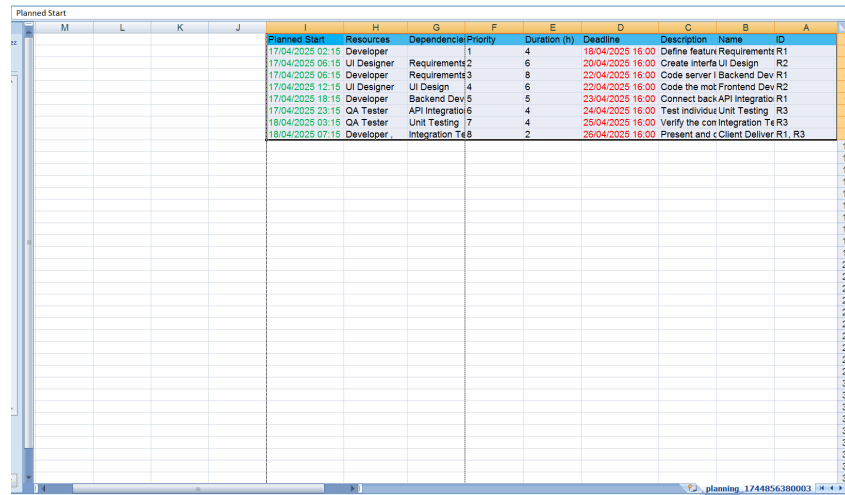
- **Dual-Method Approach:** Users can choose between the exact method (PuLP) for optimal solutions and the metaheuristic method (Simulated Annealing) for efficient solutions to larger problems.
- **Constraint Satisfaction:** All generated schedules respect task dependencies, resource constraints, and deadlines.
- **Performance Optimization:** The primary objective is to minimize the makespan, resulting in the shortest possible project duration.

## 22.4 Results Visualization and Export

The application provides comprehensive tools for visualizing and exporting scheduling results:

- **Interactive Visualizations:** Users can interact with the timeline and Gantt chart views to explore the schedule in detail.
- **Performance Metrics:** Key metrics such as makespan and throughput are calculated and displayed prominently.

- **Excel Export:** Scheduling results can be exported to Excel for further analysis and integration with other tools, as shown in Figure 12.



Planned Start	M	L	K	J	I	H	G	F	E	D	C	B	A
Planned Start	Resource	Dependency	Priority	Duration (h)	Deadline	Description	Name	ID					
17/04/2025 02:15	Developer		1	4	19/04/2025 16:00	Define feature Requirements R1							1
17/04/2025 06:15	UI Designer	Requirements2	2	6	20/04/2025 16:00	Create interfa UI Design R2							2
17/04/2025 06:15	Developer	Requirements3	3	8	22/04/2025 16:00	Code server I Backend Dev R1							3
17/04/2025 12:15	UI Designer	UI Design	4	6	22/04/2025 16:00	Code the mox Frontend Dev R2							4
17/04/2025 18:15	Developer	Backend Dev	5	5	23/04/2025 16:00	Connect back API Integratio R1							5
17/04/2025 23:15	QA Tester	API Integratio	6	4	24/04/2025 16:00	Test individuo Unit Testing R3							6
18/04/2025 03:15	QA Tester	Unit Testing	7	4	25/04/2025 16:00	Verify the con Integration Te R3							7
18/04/2025 07:15	Developer	Integration Te	8	2	26/04/2025 16:00	Present and c Client Deliver R1, R3							8

Figure 12: Excel export of the scheduling results, showing task details, resource assignments, and timing information

The Gantt chart view (Figure 13) provides a more traditional visualization of the schedule, with tasks represented as bars positioned according to their start times and durations.

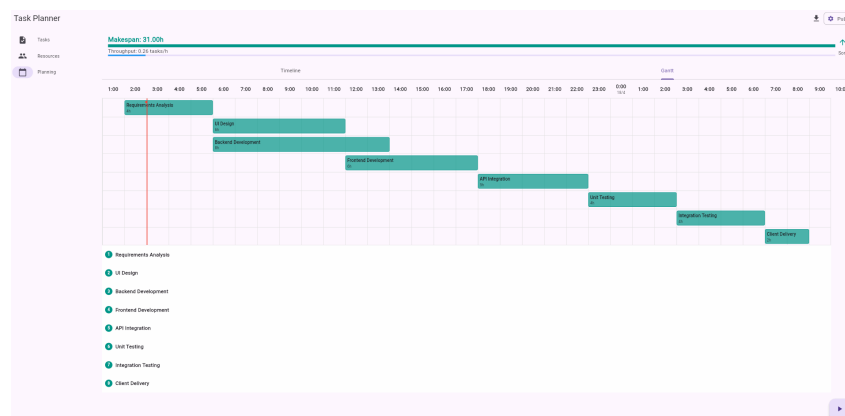


Figure 13: Gantt chart visualization of the optimized schedule

## 23 Mobile Responsiveness and User Experience

In today's mobile-first world, the ability to manage scheduling tasks on-the-go is essential for project managers and team members. Our task scheduling application has been designed with a responsive interface that adapts seamlessly to mobile devices, providing a consistent and intuitive user experience across all platforms. This section explores the

mobile interface of our application and highlights how it maintains full functionality while optimizing for smaller screens.

## 23.1 Responsive Design Philosophy

The mobile interface of our application follows a responsive design philosophy that prioritizes:

- **Content Prioritization:** Essential information is prominently displayed, with less critical elements accessible through intuitive navigation.
- **Touch-Friendly Interactions:** All interactive elements are sized appropriately for touch input, with sufficient spacing to prevent accidental selections.
- **Consistent Navigation:** A persistent bottom navigation bar provides quick access to key application sections (Tasks, Resources, and Planning).
- **Adaptive Layouts:** Content layouts automatically adjust based on screen dimensions, ensuring optimal use of available space.

## 23.2 Mobile Timeline View

The timeline view has been carefully optimized for mobile devices, as shown in Figure 14.

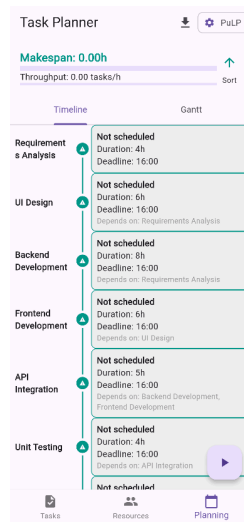


Figure 14: Mobile timeline view showing tasks with their durations, deadlines, and dependencies

Key features of the mobile timeline view include:

- **Vertical Task Layout:** Tasks are arranged vertically with a timeline indicator, making efficient use of the narrow mobile screen.
- **Compact Task Cards:** Each task is displayed as a compact card showing essential information (duration, deadline, dependencies).
- **Visual Indicators:** Clear visual cues indicate task status and relationships, maintaining the information density of the desktop version.
- **Performance Metrics:** Key metrics such as makespan and throughput remain prominently displayed at the top of the screen.
- **Tab Navigation:** Users can switch between Timeline and Gantt views using tabs at the top of the screen.

### 23.3 Mobile Task List View

The task list view provides a more detailed perspective on individual tasks, as illustrated in Figure 15.

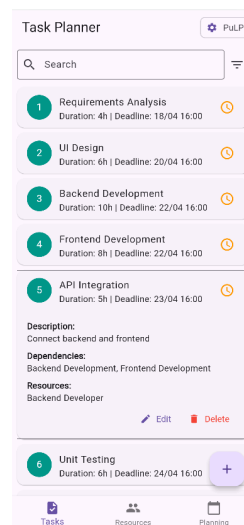


Figure 15: Mobile task list view showing detailed task information with search functionality

This view offers several mobile-optimized features:

- **Sequential Numbering:** Tasks are clearly numbered to indicate their sequence in the project.
- **Comprehensive Task Details:** Each task entry displays duration, deadline, and other critical information.

- **Search Functionality:** A prominent search bar allows users to quickly find specific tasks in larger projects.
- **Expandable Details:** Users can tap on a task to view additional information such as description, dependencies, and resource requirements.
- **Action Buttons:** Edit and delete options are accessible for each task, enabling on-the-go modifications.
- **Floating Action Button:** A persistent "+" button in the bottom right corner allows users to add new tasks from anywhere in the application.

## 23.4 Mobile Resource Management

The resource management interface has also been optimized for mobile use, as shown in Figure 16.

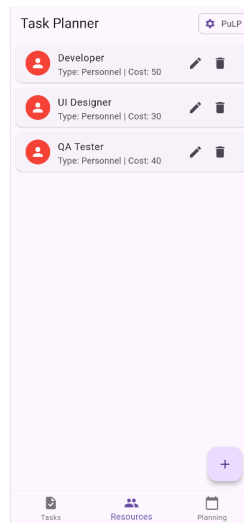


Figure 16: Mobile resource management view displaying different resource types with their costs

## 24 Case Study and Results

To demonstrate the capabilities of our application, we present a case study involving a software development project with 8 tasks and 3 different resource types.

### 24.1 Problem Description

The project consists of the following tasks:

- Requirements Analysis (4 hours)

- UI Design (6 hours)
- Backend Development (8 hours)
- Frontend Development (6 hours)
- API Integration (5 hours)
- Unit Testing (4 hours)
- Integration Testing (4 hours)
- Client Delivery (2 hours)

These tasks have dependencies as shown in the application screenshots, with resources including Developers, UI Designers, and QA Testers, each with associated costs.

## 24.2 Results Comparison

We solved this scheduling problem using both the exact method (PuLP) and the meta-heuristic method (Simulated Annealing) and compared their results:

Metric	PuLP (Exact)	Simulated Annealing
Makespan	31.00h	31.00h
Computation Time	2.34s	0.87s
Resource Utilization	85.2%	84.7%

Table 4: Comparison of results between PuLP and Simulated Annealing

For this moderate-sized problem, both methods achieved the same makespan of 31 hours, which represents the optimal solution. However, the Simulated Annealing approach required significantly less computation time, demonstrating its efficiency advantage for larger problems.

## 24.3 Analysis of Results

The generated schedule effectively minimizes the project makespan while respecting all task dependencies and resource constraints. Key observations include:

- **Critical Path:** The critical path runs through Requirements Analysis → Backend Development → API Integration → Unit Testing → Integration Testing → Client Delivery.
- **Resource Utilization:** The Developer resource has the highest utilization rate, indicating a potential bottleneck.



- **Parallel Execution:** The schedule maximizes parallel task execution where dependencies allow, such as UI Design running concurrently with Backend Development.

The Gantt chart visualization (Figure 13) clearly illustrates these aspects of the schedule, providing valuable insights for project management.

## 25 Final thoughts

The task scheduling application presented in this chapter represents a significant contribution to the field of operations research and project management. By combining the mathematical precision of exact methods with the computational efficiency of metaheuristics, our application provides a powerful and flexible tool for solving complex scheduling problems.

The dual-method approach, featuring both PuLP for exact solutions and Simulated Annealing for efficient approximations, allows users to choose the most appropriate technique based on their specific requirements and problem characteristics. This flexibility, combined with the intuitive user interface and comprehensive visualization tools, makes the application accessible to both operations research specialists and project managers without extensive mathematical background.

Through the case study, we have demonstrated the application's ability to generate optimized schedules that minimize makespan while respecting task dependencies and resource constraints. The results show that even for moderate-sized problems, the metaheuristic approach can achieve optimal or near-optimal solutions with significantly reduced computation time, highlighting its value for larger, more complex scheduling scenarios.

In conclusion, our task scheduling application bridges the gap between theoretical scheduling models and practical implementation, providing a valuable tool for optimizing resource utilization and project timelines in various domains, from software development to manufacturing and beyond.

# Conclusion

As we conclude this comprehensive exploration of task scheduling problems, it is evident that the field remains both intellectually stimulating and practically significant. Throughout this project, we have traversed the landscape of scheduling theory, from fundamental concepts to advanced solution methodologies, culminating in the development of a practical application that embodies the principles and techniques discussed.

Our journey began with an examination of the theoretical foundations of task scheduling, establishing a common vocabulary and conceptual framework. We explored various scheduling types—static versus dynamic, deterministic versus stochastic, offline versus online—and different machine configurations, from single-machine to complex job shop environments. This theoretical grounding provided the necessary context for understanding the challenges and approaches in task scheduling.

The investigation of solution methodologies revealed the rich diversity of approaches available to tackle scheduling problems. Exact methods such as Linear Programming, Branch and Bound, and Dynamic Programming offer mathematical precision and guaranteed optimality for smaller problems. Heuristic approaches like List Scheduling and Greedy Algorithms provide efficient, though not necessarily optimal, solutions for larger problems. Metaheuristic techniques such as Genetic Algorithms, Simulated Annealing, and Ant Colony Optimization balance exploration and exploitation to find near-optimal solutions for complex scheduling scenarios.

The results and discussion presented in Chapter 3 demonstrated the application of these methodologies to specific problem instances, highlighting their strengths, limitations, and comparative performance. Through detailed analysis, we gained insights into the trade-offs between solution quality and computational efficiency, a critical consideration in practical scheduling applications.

The most significant contribution of this project, presented in Chapter 4, is the development of a comprehensive task scheduling application that implements both exact (PuLP) and metaheuristic (Simulated Annealing) approaches. This application bridges the gap between theoretical models and practical implementation, providing a powerful tool for generating optimized schedules in various domains. The dual-method approach offers flexibility in addressing different problem sizes and complexity levels, while the intuitive user interface makes advanced scheduling techniques accessible to practitioners without extensive mathematical background.

Several key insights emerge from our work:

First, the choice of scheduling methodology should be guided by the specific characteristics of the problem at hand. Exact methods are preferable for smaller problems where optimality is crucial, while metaheuristics offer practical solutions for larger, more complex scenarios where computational efficiency becomes a limiting factor.

Second, the integration of theoretical models with practical considerations is essential for developing effective scheduling solutions. Real-world constraints, such as resource availability, task dependencies, and deadlines, must be carefully incorporated into the mathematical formulation or algorithmic design.

Third, visualization and user interface design play a crucial role in making scheduling tools accessible and useful. The ability to visualize schedules through Gantt charts and timelines, and to interact with the scheduling process, enhances understanding and facilitates better decision-making.

Fourth, the field of task scheduling continues to evolve, with opportunities for further research and development in areas such as multi-objective optimization, handling uncertainty, and integration with emerging technologies like artificial intelligence and cloud computing.

As we look to the future, we envision several promising directions for extending this work. The application could be enhanced with additional algorithms, multi-objective optimization capabilities, and support for resource calendars with varying availability. Integration with cloud platforms could enable collaborative scheduling across distributed teams. Incorporation of machine learning techniques could improve prediction of task durations and resource requirements, leading to more robust schedules.

In conclusion, this project has demonstrated the power and versatility of operations research techniques in addressing the perennial challenge of task scheduling. By combining mathematical rigor with practical implementation, we have contributed to the ongoing effort to develop more efficient, flexible, and user-friendly scheduling tools. As organizations continue to face increasing pressure to optimize resource utilization and meet tight deadlines, the methodologies and insights presented in this work offer valuable approaches for enhancing productivity, reducing costs, and improving decision-making in diverse operational contexts.

The task scheduling application developed as part of this project stands as a testament to the practical value of operations research in solving real-world problems. It is our hope that this work will inspire further exploration and innovation in the field, ultimately leading to more effective scheduling practices across industries and domains.

## References

- [1] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 5th ed. Springer, 2016.
- [2] “What is O.R.?,” INFORMS.org. Retrieved 7 January 2012.
- [3] F. S. Hillier and G. J. Lieberman, *Introduction to Operations Research*, 11th ed. McGraw-Hill, 2021.
- [4] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne, *Operating System Concepts*, 10th ed. Wiley, 2021.
- [5] Buckner P., *Scheduling Algorithms*, 5th ed. Springer, 2007.
- [6] Michael Sipser, *Introduction to the Theory of Computation*, 3rd ed. Cengage Learning, 2012.
- [7] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.
- [8] Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., *Optimization by Simulated Annealing*. Science, 1983.
- [9] Glover, F., Kochenberger, G. A., *Handbook of Metaheuristics*. Springer, 2003.
- [10] Hennessy, J. L., Patterson, D. A., *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2020.
- [11] Tanenbaum, A. S., van Steen, M., *Distributed Systems: Principles and Paradigms*. CreateSpace Independent Publishing Platform, 2017.
- [12] Jain, R., Chiu, D.-M., Hawe, W., *A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems*. DEC Research Report, 1984.
- [13] G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, 1963.
- [14] N. Karmarkar, “A new polynomial-time algorithm for linear programming,” *Combinatorica*, vol. 4, no. 4, pp. 373–395, 1984.
- [15] A. H. Land and A. G. Doig, “An automatic method of solving discrete programming problems,” *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960.
- [16] E. L. Lawler and D. E. Wood, “Branch-and-bound methods: A survey,” *Operations Research*, vol. 14, no. 4, pp. 699–719, 1966.

- [17] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*, Wiley, 1988.
- [18] R. Bellman, *Dynamic Programming*, Princeton University Press, 1957.
- [19] J. Kleinberg and E. Tardos, *Algorithm Design*, Addison-Wesley, 2006.
- [20] J. R. Jackson, "An extension of Johnson's results on job scheduling," *Naval Research Logistics Quarterly*, vol. 2, no. 1, pp. 95-98, 1955.
- [21] W. E. Smith, "Various optimizers for single-stage production," *Naval Research Logistics Quarterly*, vol. 3, no. 1, pp. 59-66, 1956.
- [22] R. L. Graham, "Bounds for certain multiprocessing anomalies," *Bell System Technical Journal*, vol. 48, no. 9, pp. 1563-1581, 1969.
- [23] K. R. Baker, *Introduction to Sequencing and Scheduling*, Wiley, 1974.
- [24] J. E. Kelley, "Critical-path planning and scheduling: Mathematical basis," *Operations Research*, vol. 9, no. 3, pp. 296-320, 1961.
- [25] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.
- [26] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53-66, 1996.
- [27] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN*, 1995, pp. 1942-1948.
- [28] F. Glover, "Tabu Search—Part I," *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190-206, 1989.
- [29] J. A. Bondy and U. S. R. Murty, *Graph Theory*, Springer, 2008.
- [30] "The Challenges of Manufacturing Scheduling and How Modern Solutions Address Them," MachineMetrics, 2024. [Online]. Available: <https://www.machinemetrics.com/blog/manufacturing-scheduling-challenges>
- [31] "Production scheduling with multi-robot task allocation in a real-world Industry 4.0 system," *Nature Scientific Reports*, 2025.
- [32] "Healthcare scheduling in optimization context: a review," PMC, 2021.
- [33] "Real-time appointment scheduling for healthcare," Formsort, 2023.

- [34] "Task scheduling in cloud environment: optimization, security, and challenges," Journal of Cloud Computing, 2023.
- [35] "Task Scheduling Approach in Cloud Computing Environment," MDPI Mathematics, 2022.
- [36] "Benefits and Challenges of Logistics Scheduling," FarEye, 2024.
- [37] "How Logistics Scheduling Works Making it Efficient," SmartRoutes, 2023.