# TRƯỜNG ĐẠI HỌC QUẢN LÝ VÀ CÔNG NGHỆ
# TP. HỒ CHÍ MINH

---

**BÀI BÁO CÁO ĐỒ ÁN**

**Môn học: Introduction to Artificial Intelligence**

**Sinh viên: Đinh Kim Hưng**

**Mã số sinh viên: 2302700071**

**Giảng viên: TS. Nguyễn Quản Bá Hồng**

*Thành phố Hồ Chí Minh, ngày 1 tháng 07 năm 2025*

# Mục lục

# NỘI DUNG BÁO CÁO

# LỜI MỞ ĐẦU

Trong cuộc sống hiện đại, việc quản lý thời gian và công việc cá nhân hiệu quả đóng vai trò then chốt trong việc nâng cao năng suất và đạt được mục tiêu. Tuy nhiên, việc sắp xếp một lịch trình tối ưu thường gặp nhiều thách thức do sự đa dạng về thời lượng, mức độ ưu tiên, và các mối quan hệ phụ thuộc giữa các công việc. Bài toán này trở nên phức tạp hơn khi số lượng công việc tăng lên và các ràng buộc trở nên chặt chẽ hơn.

Trí tuệ nhân tạo (AI), đặc biệt là các thuật toán heuristic, cung cấp những giải pháp mạnh mẽ để giải quyết các bài toán tối ưu hóa phức tạp như lập lịch. Các thuật toán này, dù không luôn đảm bảo tìm được lời giải tối ưu tuyệt đối, nhưng lại rất hiệu quả trong việc tìm ra các lời giải gần tối ưu trong thời gian chấp nhận được, phù hợp với tính chất động và thường thay đổi của lịch trình cá nhân.

Đồ án "Tối ưu lịch trình công việc cá nhân bằng thuật toán Heuristic" ra đời nhằm mục đích xây dựng một công cụ hỗ trợ người dùng sắp xếp công việc một cách thông minh. Chương trình sẽ tiếp nhận thông tin về các công việc (thời lượng, ưu tiên, phụ thuộc), áp dụng thuật toán heuristic để tạo ra một lịch trình không trùng lặp, ưu tiên các công việc quan trọng và tuân thủ mọi ràng buộc, đồng thời trực quan hóa kết quả để người dùng dễ dàng theo dõi.

Báo cáo này sẽ trình bày chi tiết từ cơ sở lý thuyết, phân tích bài toán, thiết kế hệ thống theo kiến trúc module, triển khai các thuật toán heuristic, cho đến kết quả đạt được, hướng dẫn sử dụng và các hướng phát triển trong tương lai.

# CHƯƠNG I: GIỚI THIỆU VÀ ĐẶT VẤN ĐỀ

## 1.1 Giới thiệu tổng quan về bài toán lập lịch

Bài toán lập lịch (scheduling problem) là một lớp bài toán tối ưu hóa quan trọng trong khoa học máy tính, nghiên cứu hoạt động và trí tuệ nhân tạo. Mục tiêu của nó là phân bổ tài nguyên có hạn (ví dụ: thời gian, nhân lực, máy móc) cho một tập hợp các nhiệm vụ (công việc) trong một khoảng thời gian nhất định, với mục đích tối ưu hóa một hoặc nhiều tiêu chí (ví dụ: giảm thiểu thời gian hoàn thành, tối đa hóa lợi nhuận, tối thiểu hóa chi phí).

Các bài toán lập lịch xuất hiện ở nhiều lĩnh vực khác nhau, từ sản xuất công nghiệp, quản lý dự án, điều phối giao thông, đến lập lịch hoạt động của máy tính và quản lý thời gian cá nhân. Sự phức tạp của bài toán thường tăng lên đáng kể khi có thêm các ràng buộc như phụ thuộc giữa các nhiệm vụ, thời gian sẵn có của tài nguyên, hoặc các mục tiêu đa dạng.

## 1.2 Mục tiêu của đồ án

Đồ án này đặt ra các mục tiêu chính sau:

- Xây dựng một chương trình quản lý lịch trình công việc cá nhân thân thiện với người dùng.
- Áp dụng thuật toán heuristic để giải quyết bài toán lập lịch công việc có ràng buộc.
- Đảm bảo các công việc trong lịch trình không bị trùng thời gian.
- Ưu tiên hoàn thành các công việc quan trọng trước.
- Tuân thủ nghiêm ngặt các ràng buộc phụ thuộc giữa các công việc.
- Tối ưu hóa tổng thời gian hoàn thành lịch trình (Makespan) và giảm thiểu khoảng trống thời gian.
- Trực quan hóa lịch trình một cách rõ ràng và dễ hiểu thông qua biểu đồ Gantt.

## 1.3 Phạm vi và các yêu cầu của đồ án

Đồ án tập trung vào việc lập lịch các công việc trong một hoặc nhiều ngày, với các đặc điểm và yêu cầu cụ thể sau:

- **Nhập liệu:** Cho phép người dùng nhập danh sách công việc bao gồm: tên công việc, khoảng thời gian thực hiện dự kiến, mức độ ưu tiên (từ 1 đến 5), và các công việc phụ thuộc (nếu có). Ngoài ra, hỗ trợ ràng buộc thời gian bắt đầu sớm nhất và kết thúc muộn nhất.
- **Sắp xếp lịch trình:** Chương trình phải sắp xếp các công việc sao cho:
  - Không có công việc nào trùng thời gian với công việc khác.
  - Các công việc có mức độ ưu tiên cao hơn được xem xét và ưu tiên hoàn thành trước.
  - Tuân thủ tuyệt đối các ràng buộc phụ thuộc (công việc X chỉ bắt đầu sau khi công việc Y hoàn thành).
- **Tối ưu hóa:** Mục tiêu là tạo ra một lịch trình tối ưu hoặc gần tối ưu về tổng thời gian hoàn thành (Makespan).
- **Xuất/Nhập dữ liệu:** Hỗ trợ lưu trữ dữ liệu công việc vào bộ nhớ cục bộ của trình duyệt (Local Storage) để duy trì trạng thái giữa các phiên làm việc, cũng như xuất/nhập dữ liệu dưới dạng file JSON.

- **Trực quan hóa:** Hiển thị lịch trình đã sắp xếp dưới dạng biểu đồ Gantt trực quan, giúp người dùng dễ dàng theo dõi.

# CHƯƠNG II: CƠ SỞ LÝ THUYẾT

## 2.1 Tổng quan về Trí tuệ nhân tạo

Trí tuệ nhân tạo (AI) là một lĩnh vực của khoa học máy tính nhằm tạo ra các hệ thống hoặc máy móc có khả năng thực hiện các chức năng nhận thức mà thông thường đòi hỏi trí thông minh của con người, như học hỏi, suy luận, giải quyết vấn đề, nhận diện mẫu, hiểu ngôn ngữ tự nhiên và tương tác. AI có thể được chia thành AI yếu (Weak AI), chuyên thực hiện một nhiệm vụ cụ thể, và AI mạnh (Strong AI), có khả năng thực hiện bất kỳ nhiệm vụ trí tuệ nào mà con người có thể làm được. Đồ án này thuộc phạm vi của AI yếu, tập trung vào việc giải quyết một bài toán tối ưu hóa cụ thể.

## 2.2 Bài toán lập lịch (Scheduling Problem)

### 2.2.1 Định nghĩa

Bài toán lập lịch là quá trình phân bổ các nguồn lực (resources) giới hạn cho một tập hợp các nhiệm vụ (tasks) trong một khoảng thời gian nhất định nhằm tối ưu hóa một hoặc nhiều tiêu chí đã định. Đây là một dạng bài toán tối ưu hóa tổ hợp, thường thuộc lớp NP-hard, nghĩa là việc tìm kiếm lời giải tối ưu tuyệt đối có thể tốn thời gian tính toán theo cấp số mũ khi kích thước bài toán tăng lên.

### 2.2.2 Các thành phần của bài toán lập lịch

Một bài toán lập lịch điển hình bao gồm các thành phần sau:

- **Nhiệm vụ (Tasks/Jobs):** Các hoạt động cần được thực hiện. Mỗi nhiệm vụ có thể có các thuộc tính như thời lượng, thời gian đến, thời hạn, mức độ ưu tiên.
- **Tài nguyên (Resources):** Các nguồn lực có sẵn để thực hiện nhiệm vụ (ví dụ: máy móc, nhân lực, thời gian). Tài nguyên có thể là duy nhất hoặc chia sẻ.
- **Ràng buộc (Constraints):** Các điều kiện mà việc lập lịch phải tuân thủ (ví dụ: một nhiệm vụ phải hoàn thành trước nhiệm vụ khác, một tài nguyên không thể thực hiện hai nhiệm vụ cùng lúc).

- **Mục tiêu (Objectives):** Các tiêu chí để đánh giá "chất lượng" của một lịch trình (ví dụ: tối thiểu hóa thời gian hoàn thành tổng thể, tối đa hóa sử dụng tài nguyên, tối thiểu hóa độ trễ).

### 2.2.3 Các ràng buộc phổ biến trong lập lịch

- **Ràng buộc thời gian (Temporal Constraints):**
  - **Phụ thuộc (Precedence Constraints):** Nhiệm vụ A phải hoàn thành trước khi nhiệm vụ B có thể bắt đầu.
  - **Thời gian đến (Release Dates):** Nhiệm vụ không thể bắt đầu trước một thời điểm nhất định.
  - **Thời hạn (Deadlines):** Nhiệm vụ phải hoàn thành trước một thời điểm nhất định.
- **Ràng buộc tài nguyên (Resource Constraints):**
  - **Độc quyền sử dụng:** Một tài nguyên chỉ có thể được sử dụng bởi một nhiệm vụ tại một thời điểm.
  - **Khả năng đáp ứng:** Tài nguyên có thể có giới hạn về khả năng thực hiện nhiệm vụ.

### 2.2.4 Mục tiêu tối ưu trong lập lịch

Tùy thuộc vào bài toán, có nhiều mục tiêu tối ưu khác nhau:

- **Makespan:** Tổng thời gian hoàn thành của tất cả các nhiệm vụ. Mục tiêu thường là tối thiểu hóa Makespan.
- **Total Lateness/Tardiness:** Tổng độ trễ hoặc độ trễ tích lũy của các nhiệm vụ so với thời hạn của chúng.
- **Resource Utilization:** Tỷ lệ tài nguyên được sử dụng hiệu quả.
- **Priority Fulfillment:** Đảm bảo các nhiệm vụ quan trọng được ưu tiên hoàn thành.

## 2.3 Thuật toán Heuristic

### 2.3.1 Khái niệm Heuristic

Trong khoa học máy tính và trí tuệ nhân tạo, heuristic là một kỹ thuật, một phương pháp hoặc một quy tắc chỉ dẫn được thiết kế để giải quyết một bài toán một cách nhanh chóng hơn so với các phương pháp chính xác (optimal) hoặc để tìm ra một lời giải chấp nhận được khi việc tìm lời giải tối ưu là không khả thi về mặt tính toán. Heuristic dựa trên kinh nghiệm hoặc trực giác để đưa ra "dự đoán tốt nhất" cho một bước đi cụ thể trong quá trình tìm kiếm.

Đặc điểm của heuristic:

- **Không đảm bảo tối ưu:** Heuristic không đảm bảo tìm được lời giải tối ưu tuyệt đối.
- **Hiệu quả về thời gian:** Thường nhanh hơn và ít tốn tài nguyên hơn các thuật toán chính xác.
- **Tìm giải pháp chấp nhận được:** Hữu ích cho các bài toán phức tạp, nơi lời giải tối ưu quá đắt đỏ để tìm kiếm.

### 2.3.2 Vai trò của Heuristic trong tìm kiếm và tối ưu

Heuristic đóng vai trò quan trọng trong các thuật toán tìm kiếm có thông tin (informed search algorithms) như A*, Best-First Search, và trong các thuật toán tối ưu hóa như Greedy, Simulated Annealing, Tabu Search, Genetic Algorithms. Chúng giúp hướng dẫn quá trình tìm kiếm hoặc xây dựng giải pháp bằng cách đánh giá "sự hứa hẹn" của một lựa chọn hoặc trạng thái, từ đó giảm đáng kể không gian tìm kiếm.

### 2.3.3 Phân loại các thuật toán Heuristic

- **Thuật toán Greedy (Tham lam):**
  - **Ý tưởng:** Ở mỗi bước, thuật toán đưa ra lựa chọn tốt nhất cục bộ (local optimum) với hy vọng rằng chuỗi các lựa chọn cục bộ này sẽ dẫn đến một lời giải tốt toàn cục.
  - **Đặc điểm:** Đơn giản, nhanh chóng, nhưng không đảm bảo lời giải tối ưu.
  - **Ứng dụng:** Thường được dùng làm phương pháp khởi tạo hoặc cho các bài toán mà lời giải tối ưu không quá quan trọng.
- **Thuật toán A* (A-star Algorithm):**
  - **Ý tưởng:** Là một thuật toán tìm kiếm đường đi tốt nhất (Best-First Search) có thông tin, sử dụng một hàm đánh giá f(n) = g(n) + h(n).
    - g(n): Chi phí thực tế từ điểm bắt đầu đến nút n.
    - h(n): Ước lượng chi phí (heuristic) từ nút n đến mục tiêu.
  - **Đặc điểm:** Đảm bảo tìm được lời giải tối ưu nếu hàm heuristic là "consistent" (tức là thỏa mãn tính chất đơn điệu) và admissible (không bao giờ ước lượng quá cao chi phí thực tế). Tuy nhiên, có thể tốn nhiều bộ nhớ.
  - **Ứng dụng:** Tìm đường đi ngắn nhất, lập kế hoạch, giải các bài toán như 8-puzzle.

# CHƯƠNG III: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

## 3.1 Phân tích bài toán "Tối ưu lịch trình công việc cá nhân"

Bài toán này được mô hình hóa như một bài toán lập lịch có ràng buộc, với các đặc điểm cụ thể của công việc cá nhân.

### 3.1.1 Các yếu tố của một công việc (Task)

Mỗi công việc được định nghĩa bằng các thuộc tính sau:

- id (string): Mã định danh duy nhất (ví dụ: "T1", "T2").
- name (string): Tên mô tả công việc.
- duration (number): Thời lượng dự kiến để hoàn thành công việc (đơn vị phút).
- priority (number): Mức độ ưu tiên từ 1 (thấp nhất) đến 5 (cao nhất).
- dependencies (string[]): Một mảng các id công việc khác mà công việc này phải hoàn thành sau chúng.
- earliestStartTime (Date | null): Ràng buộc thời gian bắt đầu sớm nhất.
- latestFinishTime (Date | null): Ràng buộc thời gian kết thúc muộn nhất.
- actualStartTime (Date | null): Thời gian bắt đầu thực tế sau khi lập lịch.
- actualEndTime (Date | null): Thời gian kết thúc thực tế sau khi lập lịch.
- status (string): Trạng thái của công việc ("pending", "scheduled").

### 3.1.2 Các ràng buộc cụ thể trong đồ án

- **Không trùng thời gian:** Không có hai công việc nào được thực hiện cùng một khoảng thời gian.
- **Tuân thủ phụ thuộc:** Một công việc chỉ có thể bắt đầu khi tất cả các công việc trong danh sách dependencies của nó đã kết thúc.
- **Ràng buộc thời gian sớm nhất/muộn nhất:** Nếu earliestStartTime được định nghĩa, công việc không thể bắt đầu trước thời điểm đó. Nếu latestFinishTime được định nghĩa, công việc phải kết thúc trước thời điểm đó.

### 3.1.3 Mục tiêu tối ưu của đồ án

- **Ưu tiên cao hơn được hoàn thành trước:** Đây là mục tiêu chính, được tích hợp vào hàm heuristic.
- **Tối thiểu hóa Makespan:** Giảm thiểu tổng thời gian từ khi công việc đầu tiên bắt đầu đến khi công việc cuối cùng kết thúc.

- **Tối thiểu hóa khoảng trống thời gian:** Ưu tiên lấp đầy các khoảng trống trống nếu có thể.

## 3.2 Mô hình hóa dữ liệu

### 3.2.1 Lớp Task và các thuộc tính

Lớp Task trong scripts/models/Task.js là thành phần cốt lõi để biểu diễn thông tin về một công việc.

```JavaScript
// scripts/models/Task.js
class Task {
    constructor(id, name, duration, priority, dependencies = [], earliestStartTime = null, latestFinishTime = null) {
        // ... (Logic kiểm tra input)
        this.id = id;
        this.name = name;
        this.duration = duration; // In minutes
        this.priority = priority; // 1 (low) to 5 (high)
        this.dependencies = dependencies; // Array of task IDs
        this.actualStartTime = null;
        this.actualEndTime = null;
        this.earliestStartTime = earliestStartTime;
        this.latestFinishTime = latestFinishTime;
        this.status = "pending";
    }
    // ... (isScheduled, setSchedule, toObject, fromObject methods)
}
```

Lớp này bao gồm các hàm tiện ích như isScheduled() để kiểm tra trạng thái lập lịch, setSchedule() để gán thời gian thực tế, toObject() và fromObject() để chuyển đổi giữa đối tượng Task và plain object JSON, phục vụ cho việc lưu trữ/tải dữ liệu.

### 3.2.2 Các tiện ích dữ liệu (Validation, DateUtils)

- **Validation.js**: Cung cấp các hàm để kiểm tra tính hợp lệ của dữ liệu đầu vào từ người dùng, bao gồm isValidDuration(), isValidPriority(), isValidDependency(), isNonEmptyString(), isValidDate(). Điều này đảm bảo dữ liệu luôn sạch trước khi được xử lý bởi thuật toán.
- **DateUtils.js**: Chứa các hàm hỗ trợ thao tác với đối tượng Date của JavaScript, như addMinutes(), addHours(), diffInMinutes(), và formatDateTime(). Các hàm này rất quan trọng cho việc tính toán thời gian, kiểm tra ràng buộc và hiển thị lịch trình đa ngày.

## 3.3 Thiết kế kiến trúc tổng thể (Modular Design)

Hệ thống được thiết kế theo kiến trúc module hóa với 5 module chính, giúp phân tách rõ ràng các trách nhiệm, tăng tính dễ bảo trì, dễ mở rộng và dễ kiểm thử.

- **Module 1: Core Data Models & Utilities:** Định nghĩa cấu trúc dữ liệu (Task) và các hàm tiện ích chung (Validation, DateUtils).
- **Module 2: User Interface (UI) - Nhập liệu & Điều khiển:** Quản lý giao diện nhập liệu công việc và hiển thị danh sách công việc (TaskFormHandler).
- **Module 3: Scheduling Algorithms:** Chứa logic cốt lõi của thuật toán heuristic để lập lịch (HeuristicScheduler, Heuristics).
- **Module 4: Visualization & Output:** Xử lý việc hiển thị lịch trình đã tính toán một cách trực quan (ScheduleRenderer, GanttChart).
- **Module 5: Main Application & Orchestration:** Là điểm vào của ứng dụng, điều phối luồng dữ liệu, tương tác giữa các module và quản lý lưu/tải dữ liệu (main.js, DataLoader).

## 3.4 Thiết kế thuật toán Heuristic Scheduler

### 3.4.1 Lựa chọn thuật toán chính (Greedy Approach)

Trong giai đoạn đầu của đồ án, thuật toán Greedy đã được lựa chọn làm phương pháp lập lịch chính. Lý do cho lựa chọn này bao gồm:

- **Tính đơn giản:** Thuật toán Greedy dễ hiểu và triển khai, phù hợp cho việc xây dựng một phiên bản chức năng cơ bản nhanh chóng.
- **Hiệu suất chấp nhận được:** Đối với các bài toán lập lịch quy mô nhỏ đến trung bình, Greedy thường mang lại kết quả chấp nhận được trong thời gian tính toán ngắn.

- Khả năng mở rộng: Kiến trúc module cho phép dễ dàng thay thế hoặc kết hợp Greedy với các thuật toán phức tạp hơn (như A* hoặc Local Search) trong tương lai. Thuật toán Greedy hoạt động bằng cách tại mỗi bước, nó sẽ chọn công việc "tốt nhất" theo một tiêu chí heuristic đã định, sau đó gán công việc đó vào vị trí sớm nhất có thể mà không vi phạm các ràng buộc, rồi tiếp tục với các công việc còn lại.

### 3.4.2 Các hàm Heuristic cụ thể được sử dụng

Các hàm heuristic này được định nghĩa trong scripts/algorithms/heuristics.js và đóng vai trò quan trọng trong việc hướng dẫn thuật toán Greedy lựa chọn công việc nào sẽ được lập lịch tiếp theo. Giá trị trả về càng cao thường được ưu tiên hơn.

- **Heuristic theo ưu tiên (byPriority):**
  JavaScript
  ```
  byPriority: (task) => { return task.priority; }
  ```

  Công việc có mức độ ưu tiên cao hơn (ví dụ: 5 là cao nhất) sẽ có giá trị heuristic cao hơn và do đó được ưu tiên lập lịch trước.
- **Heuristic theo thời lượng (byDuration):**
  JavaScript
  ```
  byDuration: (task) => { return -task.duration; } // Dùng âm để ưu tiên thời lượng ngắn hơn
  ```

  Đây là heuristic "Shortest Job First" (SJF). Công việc có thời lượng ngắn hơn sẽ có giá trị âm ít hơn, tức là giá trị heuristic cao hơn, và được ưu tiên lập lịch trước.
- **Heuristic kết hợp (combined):**
  JavaScript
  ```
  combined: (task, factors = { priority: 1, duration: 0.1 }) => {
    return (task.priority * factors.priority) - (task.duration * factors.duration);
  }
  ```

  Đây là heuristic chính được sử dụng trong đồ án. Nó kết hợp mức độ ưu tiên và thời lượng của công việc. Bằng cách điều chỉnh factors.priority và factors.duration, có thể kiểm soát mức độ ảnh hưởng của từng yếu tố. Ví dụ, priority: 10, duration: 0.01 cho thấy ưu tiên mức độ quan trọng cao hơn nhiều so với việc hoàn thành công việc nhanh.
- **Heuristic theo thời gian bắt đầu sớm nhất (byEarliestStartTime):**
  JavaScript

```
byEarliestStartTime: (task) => {
        return task.earliestStartTime ? -task.earliestStartTime.getTime() :
Number.MIN_SAFE_INTEGER;
} // Ưu tiên thời gian sớm hơn (giá trị timestamp nhỏ hơn)
```

Heuristic này được thiết kế để ưu tiên các công việc có ràng buộc earliestStartTime sớm hơn. Nếu một công việc không có ràng buộc này, nó sẽ có giá trị heuristic rất thấp, khiến nó ít được ưu tiên bởi heuristic này.

### 3.4.3 Chi tiết thuật toán HeuristicScheduler

Thuật toán được triển khai trong lớp HeuristicScheduler (scripts/algorithms/HeuristicScheduler.js).

3.4.3.1 Các bước thực hiện

1. **Sao chép dữ liệu:** Tạo một bản sao sâu của mảng Task đầu vào để tránh thay đổi trực tiếp dữ liệu gốc.
2. **Khởi tạo:**
   - scheduledTasks: Mảng rỗng để chứa các công việc đã được lập lịch.
   - pendingTasks: Mảng chứa tất cả các công việc chưa được lập lịch.
   - taskMap: Một Map để tra cứu công việc theo ID một cách nhanh chóng.
3. **Vòng lặp lập lịch:**
   - Lặp cho đến khi pendingTasks rỗng hoặc không còn công việc nào có thể lập lịch được (schedulableTasks rỗng).
   - **Lọc công việc có thể lập lịch (schedulableTasks):** Chỉ những công việc mà tất cả các phụ thuộc của chúng đã được lập lịch mới được xem xét.
   - **Tìm công việc tốt nhất:** Duyệt qua schedulableTasks, tính toán giá trị heuristic cho từng công việc bằng heuristicFn đã chọn. Đồng thời, tính bestPossibleStartTime cho mỗi công việc.
   - **Chọn bestTaskToSchedule:** Chọn công việc có giá trị heuristic cao nhất. Nếu có nhiều công việc có cùng heuristic cao nhất, ưu tiên công việc có bestPossibleStartTime sớm nhất để tối ưu khoảng trống.
   - **Lên lịch:** Gán actualStartTime và actualEndTime cho bestTaskToSchedule và thêm nó vào scheduledTasks. Xóa nó khỏi pendingTasks.
4. **Sắp xếp cuối cùng:** Sau khi vòng lặp kết thúc, scheduledTasks được sắp xếp lại theo actualStartTime để trình bày rõ ràng.

3.4.3.2 Cách xử lý ràng buộc phụ thuộc

Trước khi một công việc được xem xét để lập lịch, thuật toán kiểm tra rằng TẤT CẢ các công việc trong mảng dependencies của nó phải có actualEndTime được xác định (tức là đã được isScheduled()).
potentialStartTime của một công việc sẽ được tính toán để đảm bảo nó không bắt đầu trước actualEndTime của bất kỳ công việc phụ thuộc nào.

3.4.3.3 Cách xử lý chồng chéo thời gian

Đây là một phần quan trọng để đảm bảo không có công việc nào trùng lặp. Khi tìm bestPossibleStartTime cho một công việc:

1. Bắt đầu với thời gian sớm nhất có thể dựa trên phụ thuộc và earliestStartTime.
2. Duyệt qua tất cả scheduledTasks.
3. Nếu khoảng thời gian của công việc hiện tại bị chồng chéo với một scheduledTask nào đó, potentialStartTime của công việc hiện tại sẽ được đẩy lên sau actualEndTime của scheduledTask đó.
4. Quá trình này lặp lại cho đến khi tìm được một khoảng trống không bị chồng chéo với bất kỳ công việc nào đã được lập lịch.

3.4.3.4 Cách xử lý ràng buộc thời gian bắt đầu/kết thúc sớm nhất/muộn nhất

- earliestStartTime: Nếu task.earliestStartTime tồn tại và lớn hơn potentialStartTime hiện tại (tính từ phụ thuộc), thì potentialStartTime sẽ được cập nhật bằng task.earliestStartTime.
- latestFinishTime: Sau khi tìm được currentConsideredStartTime không bị chồng chéo, thuật toán kiểm tra xem currentConsideredStartTime cộng với duration có vượt quá task.latestFinishTime không. Nếu có, công việc đó sẽ bị bỏ qua ở bước hiện tại hoặc có thể được đánh dấu là không thể lập lịch.

## 3.5 Thiết kế giao diện và trực quan hóa

### 3.5.1 Cấu trúc giao diện HTML

Giao diện người dùng được xây dựng trên một tệp index.html duy nhất, chia thành các section rõ ràng:

- **Header:** Tiêu đề ứng dụng.
- **Task Input Section:** Chứa form để nhập thông tin công việc mới.

- **Task List Section:** Hiển thị danh sách các công việc đã nhập dưới dạng bảng, kèm các nút quản lý (Thêm, Sửa, Xóa).
- **Schedule Output Section:** Hiển thị tổng thời gian hoàn thành (Makespan) và biểu đồ Gantt.
- **Footer:** Thông tin bản quyền.

### 3.5.2 Thiết kế CSS (bố cục, màu sắc, responsive)

Các tệp CSS (main.css, components/task-form.css, components/task-list.css, components/schedule-display.css) được sử dụng để:

- Thiết lập bố cục tổng thể trang (sử dụng Flexbox/Grid để sắp xếp các form và section).
- Tạo kiểu cho các thành phần UI (nút bấm, trường nhập liệu, bảng) với màu sắc, font chữ và hiệu ứng hover nhất quán.
- Đảm bảo tính phản hồi (responsive design) để giao diện hiển thị tốt trên các kích thước màn hình khác nhau.
- Đặc biệt tập trung vào việc cải thiện độ tương phản và dễ nhìn cho biểu đồ Gantt.

### 3.5.3 Thư viện trực quan hóa (Frappe Gantt)

Ban đầu, Google Charts được sử dụng, nhưng do hạn chế về khả năng tùy chỉnh màu sắc và UI/UX (đặc biệt là độ tương phản của chữ và màu nền thanh), đồ án đã chuyển sang sử dụng thư viện Frappe Gantt.

Ưu điểm của Frappe Gantt:

- **Trực quan cao:** Biểu đồ đẹp mắt, dễ hiểu ngay từ cài đặt mặc định.
- **Khả năng tùy chỉnh:** Cho phép tùy chỉnh màu sắc chi tiết hơn thông qua CSS custom_class.
- **Hỗ trợ tương tác:** Dễ dàng triển khai kéo thả, thay đổi thời lượng (tính năng nâng cao cho tương lai).
- **Dễ sử dụng:** API đơn giản và dễ tích hợp.

### 3.5.4 Cách ánh xạ dữ liệu sang Frappe Gantt

Dữ liệu từ các đối tượng Task của đồ án được chuyển đổi sang định dạng mà Frappe Gantt yêu cầu:

JavaScript

```
const frappeTasks = scheduledTasks.map(task => {
  // ...
```

```
    return {
        id: task.id,
        name: task.name,
        start: task.actualStartTime.toISOString().split('T')[0], // YYYY-MM-DD
        end: task.actualEndTime.toISOString().split('T')[0],     // YYYY-MM-DD
        progress: 100,
        dependencies: task.dependencies.join(','),
        custom_class: `priority-${task.priority}` // Dùng để CSS màu sắc theo ưu tiên
    };
});
```

Mỗi mức độ ưu tiên (Priority 1-5) được ánh xạ tới một custom_class riêng, cho phép CSS định nghĩa màu sắc cho từng loại thanh công việc.

### 3.5.5 Các tùy chỉnh UI/UX cho biểu đồ Gantt

- **Màu sắc thanh công việc:** Sử dụng các lớp CSS như .gantt-bar.priority-X .bar-progress để áp dụng màu sắc cụ thể cho từng mức độ ưu tiên, đảm bảo sự phân biệt rõ ràng và tương phản cao.
- **Màu chữ trong biểu đồ:** Đặt fill cho các class CSS của Frappe Gantt (như .gantt-bar-label, .gantt-left .gantt-row-label, .gantt-top .gantt-column-label) thành màu xám đậm (#333 hoặc #555) để đảm bảo tất cả text đều dễ đọc trên nền sáng.
- **Kích thước và khoảng cách thanh:** Tùy chỉnh bar_height, bar_padding, padding trong cấu hình Frappe Gantt để các thanh có kích thước hợp lý và dễ nhìn.
- **Tooltip tùy chỉnh:** Sử dụng hàm popup_html của Frappe Gantt để tạo ra tooltip hiển thị đầy đủ thông tin chi tiết về công việc (ID, tên, thời lượng, ưu tiên, thời gian, phụ thuộc, ràng buộc thời gian) với định dạng và màu sắc rõ ràng (nền tối, chữ sáng).
- **Thanh timeline:** Tùy chỉnh view_mode, column_width của Frappe Gantt để timeline hiển thị hợp lý hơn, dễ theo dõi các mốc thời gian.

# CHƯƠNG IV: TRIỂN KHAI VÀ KẾT QUẢ

## 4.1 Môi trường phát triển và công nghệ sử dụng

- **Ngôn ngữ lập trình:** JavaScript (cho logic chính), HTML (cho cấu trúc giao diện), CSS (cho tạo kiểu).
- **Môi trường chạy:** Trình duyệt web (Chrome, Firefox, Edge, v.v.).

- **Thư viện/Framework:**
  - **Frappe Gantt:** Thư viện JavaScript chuyên biệt cho biểu đồ Gantt.
  - (Có thể thêm) Bootstrap/TailwindCSS nếu có sử dụng framework CSS.
- **Công cụ phát triển:** Visual Studio Code, Developer Tools của trình duyệt.

## 4.2 Cấu trúc thư mục dự án

```
personal-task-scheduler/
├── index.html
├── styles/
│   ├── main.css
│   └── components/
│       ├── task-form.css
│       ├── task-list.css
│       └── schedule-display.css
├── scripts/
│   ├── main.js
│   ├── models/
│   │   └── Task.js
│   ├── algorithms/
│   │   ├── HeuristicScheduler.js
│   │   └── heuristics.js
│   ├── utils/
│   │   ├── validation.js
│   │   ├── dateUtils.js
│   │   └── dataLoader.js
│   ├── ui/
│   │   ├── taskFormHandler.js
│   │   └── scheduleRenderer.js
│   └── visualizations/
│       └── ganttChart.js
└── data/
    └── sample-tasks.json
```

Mỗi thư mục và tệp tin được tổ chức theo trách nhiệm module đã định, đảm bảo tính rõ ràng và dễ quản lý trong quá trình phát triển.

## 4.3 Các chức năng chính đã triển khai

### 4.3.1 Nhập liệu và quản lý công việc (Thêm, Sửa, Xóa, Xóa tất cả)

- **Thêm công việc:** Giao diện form cho phép nhập tên, thời lượng, mức độ ưu tiên, phụ thuộc, thời gian bắt đầu/kết thúc sớm/muộn nhất. Dữ liệu được kiểm tra hợp lệ trước khi tạo đối tượng Task.
- **Sửa công việc:** Nút "Sửa" trên mỗi hàng công việc trong bảng. Khi nhấn, dữ liệu của công việc đó được tải lại vào form để chỉnh sửa. Sau khi chỉnh sửa, trạng thái của công việc được đặt lại về "pending" để thuật toán có thể lập lịch lại.
- **Xóa công việc:** Nút "Xóa" trên mỗi hàng. Chương trình kiểm tra ràng buộc phụ thuộc: không cho phép xóa một công việc nếu có công việc khác đang phụ thuộc vào nó.
- **Xóa tất cả công việc:** Nút riêng biệt để xóa toàn bộ danh sách công việc khỏi bảng và bộ nhớ.

### 4.3.2 Chức năng lưu/tải/xuất/nhập dữ liệu công việc

- **Lưu tự động:** Mọi thay đổi trong danh sách công việc (thêm, sửa, xóa, lập lịch) đều được tự động lưu vào Local Storage của trình duyệt. Điều này đảm bảo dữ liệu không bị mất khi đóng ứng dụng hoặc làm mới trang.
- **Tải tự động:** Khi ứng dụng khởi động, dữ liệu công việc sẽ được tự động tải từ Local Storage và hiển thị trên giao diện.
- **Xuất dữ liệu:** Nút "Xuất công việc" cho phép người dùng tải xuống một file JSON chứa toàn bộ danh sách công việc hiện tại.
- **Nhập dữ liệu:** Nút "Nhập công việc" cho phép người dùng tải lên một file JSON (có định dạng tương thích) để khôi phục hoặc thêm một danh sách công việc.

### 4.3.3 Chức năng lập lịch công việc

- Khi người dùng nhấn nút "Sắp xếp lịch trình", thuật toán HeuristicScheduler sẽ được kích hoạt.
- Thuật toán sẽ nhận danh sách công việc hiện có, áp dụng heuristic đã chọn (hiện tại là combined priority và duration), và sắp xếp chúng dựa trên các ràng buộc phụ

thuộc, tránh trùng lặp thời gian, và tuân thủ các ràng buộc thời gian bắt đầu/kết thúc sớm/muộn nhất.
- Kết quả là một danh sách các công việc đã được gán thời gian actualStartTime và actualEndTime.
- Trạng thái của các công việc trong bảng danh sách sẽ được cập nhật thành "Đã lên lịch".

### 4.3.4 Trực quan hóa lịch trình bằng biểu đồ Gantt

- Sau khi lịch trình được tính toán, ScheduleRenderer sẽ nhận kết quả và sử dụng thư viện Frappe Gantt để vẽ biểu đồ.
- Biểu đồ hiển thị các công việc dưới dạng các thanh ngang trên trục thời gian, với vị trí và độ dài thanh tương ứng với thời gian bắt đầu và thời lượng.
- Màu sắc của các thanh công việc được tùy chỉnh dựa trên mức độ ưu tiên, giúp người dùng dễ dàng nhận biết công việc quan trọng.
- Tổng thời gian hoàn thành (Makespan) của lịch trình cũng được hiển thị rõ ràng.
- Tooltip tùy chỉnh hiển thị đầy đủ thông tin chi tiết của công việc khi di chuột qua.

## 4.4 Hướng dẫn sử dụng chương trình

4.4.1 Khởi động ứng dụng

Để sử dụng chương trình, người dùng chỉ cần mở tệp index.html trong bất kỳ trình duyệt web hiện đại nào (Chrome, Firefox, Edge, Safari).

### 4.4.2 Thêm công việc

- Truy cập mục "Nhập công việc mới".
- Điền các thông tin: "Tên công việc", "Thời lượng (phút)", "Mức độ ưu tiên (1-5)".
- Đối với "Phụ thuộc", nhập ID của công việc mà công việc này cần hoàn thành sau đó (ví dụ: T1, T3 nếu công việc này phụ thuộc vào công việc có ID là T1 và T3).
- (Tùy chọn) Điền "Bắt đầu sớm nhất" và "Kết thúc muộn nhất" nếu có ràng buộc thời gian cụ thể.
- Nhấn nút "Thêm công việc". Công việc sẽ xuất hiện trong "Danh sách công việc" bên dưới.

### 4.4.3 Sửa/Xóa công việc

- Trong "Danh sách công việc", tìm công việc muốn thao tác.

- Nhấn nút "Sửa" tương ứng với công việc để điền thông tin của nó vào form nhập liệu, sau đó chỉnh sửa và nhấn "Cập nhật công việc".
- Nhấn nút "Xóa" để xóa công việc. Nếu công việc có các phụ thuộc chưa được giải quyết, hệ thống sẽ báo lỗi và không cho xóa.
- Nhấn nút "Xóa tất cả" để xóa toàn bộ danh sách công việc.

#### 4.4.4 Lập lịch và xem kết quả

- Sau khi thêm đủ các công việc, nhấn nút "Sắp xếp lịch trình".
- Hệ thống sẽ chạy thuật toán và hiển thị "Tổng thời gian hoàn thành (Makespan)" và biểu đồ Gantt trong mục "Lịch trình đã sắp xếp".
- Di chuột qua các thanh công việc trong biểu đồ Gantt để xem thông tin chi tiết (tooltip).

#### 4.4.5 Lưu/Tải/Xuất/Nhập dữ liệu

- Dữ liệu công việc được tự động lưu vào trình duyệt sau mỗi thao tác.
- Sử dụng nút "Xuất công việc" để tải xuống file JSON chứa dữ liệu.
- Sử dụng nút "Nhập công việc" để chọn file JSON đã xuất trước đó và tải dữ liệu vào ứng dụng.

### 4.5 Đánh giá kết quả và hạn chế

#### 4.5.1 Ưu điểm của phương pháp Heuristic đã chọn

- **Hiệu quả và nhanh chóng:** Thuật toán Greedy cho phép tạo ra lịch trình trong thời gian ngắn, ngay cả với số lượng công việc tương đối lớn, phù hợp với ứng dụng cá nhân.
- **Đáp ứng ràng buộc cốt lõi:** Đảm bảo không trùng thời gian và tuân thủ phụ thuộc một cách chặt chẽ.
- **Trực quan hóa hiệu quả:** Việc sử dụng Frappe Gantt đã cải thiện đáng kể khả năng theo dõi và hiểu lịch trình của người dùng, với màu sắc phân biệt ưu tiên và tooltip chi tiết.
- **Linh hoạt trong tùy chỉnh:** Các hàm heuristic có thể dễ dàng thay đổi hoặc kết hợp với các trọng số khác nhau để phù hợp với các tiêu chí ưu tiên khác nhau của người dùng.

### 4.5.2 Hiệu quả của lịch trình được tạo ra

Lịch trình được tạo ra là một giải pháp gần tối ưu dựa trên tiêu chí heuristic đã chọn. Nó ưu tiên các công việc quan trọng và lấp đầy khoảng trống có sẵn. Đối với bài toán lập lịch công việc cá nhân, việc tìm kiếm một giải pháp tối ưu tuyệt đối (thường tốn kém về mặt tính toán) ít quan trọng hơn việc có một lịch trình hoạt động, hợp lý và có thể áp dụng được trong thực tế.

### 4.5.3 Các hạn chế hiện tại của thuật toán và UI/UX

- **Không đảm bảo tối ưu toàn cục:** Thuật toán Greedy chỉ đưa ra các quyết định tốt nhất cục bộ, có thể bỏ lỡ một lịch trình tổng thể tốt hơn.
- **Thiếu xử lý vòng lặp phụ thuộc:** Hiện tại, thuật toán chưa có cơ chế phát hiện và cảnh báo vòng lặp phụ thuộc (ví dụ: A phụ thuộc B, B phụ thuộc A), dẫn đến việc các công việc đó sẽ không bao giờ được lập lịch.
- **Ràng buộc tài nguyên hạn chế:** Chương trình chưa xem xét các ràng buộc tài nguyên khác ngoài thời gian (ví dụ: có nhiều người thực hiện công việc, mỗi người chỉ có thể làm 1 việc).
- **UI/UX cơ bản:** Mặc dù đã cải thiện, giao diện vẫn chưa có tính năng kéo thả trực tiếp trên biểu đồ Gantt để điều chỉnh lịch trình thủ công, hoặc các thông báo người dùng chưa thực sự thông minh (vẫn dùng alert()).
- **Tính toán Makespan đơn giản:** Chỉ tính tổng thời gian làm việc, chưa tính khoảng trống thời gian.

### 4.5.4 Thách thức đã gặp phải trong quá trình triển khai

- **Xử lý định dạng Date/Time:** Thách thức ban đầu với Google Charts trong việc đảm bảo kiểu dữ liệu Date hợp lệ và định dạng phù hợp.
- **Chuyển đổi thư viện Gantt:** Việc chuyển từ Google Charts sang Frappe Gantt đòi hỏi việc ánh xạ lại cấu trúc dữ liệu và điều chỉnh logic vẽ biểu đồ.
- **Kiểm tra ràng buộc phức tạp:** Đảm bảo thuật toán xử lý đúng tất cả các ràng buộc phụ thuộc, tránh chồng chéo, và tuân thủ earliestStartTime/latestFinishTime.
- **Quản lý trạng thái ứng dụng:** Đảm bảo dữ liệu trong TaskFormHandler luôn đồng bộ với dữ liệu được Scheduler xử lý và Visualizer hiển thị.