Đại Học Khoa Học Tự Nhiên

KHOA Toán-Cơ-Tin học

BÁO CÁO GIỮA KỲ

Implementation Plan Report

Môn học: Xây dựng phần mềm

Giảng viên hướng dẫn: Hoàng Anh Đức

Thành viên:

- Tạ Văn Sơn – MSSV: 22000124
- Nguyễn Công An – MSSV: 22000068
- Mai Văn Phương – MSSV: 22000115
- Đào Thanh Quang – MSSV: 22000118
- Trần Hoàng Đức – MSSV: 22000089
- Phùng Hữu Uy– MSSV: 22001654

Hà Nội, 2026

Mục lục

1 Tổng quan và mục tiêu hệ thống

Dự án hướng tới việc xây dựng một ứng dụng web tương tác (webapp) có khả năng **giải thích cơ chế hoạt động** của solver PARIS trong bài toán Independent Set Reconfiguration (ISR). Không giống các hệ thống chỉ cung cấp kết quả đầu ra, hệ thống này tập trung vào việc minh bạch hóa toàn bộ pipeline: từ biểu diễn bài toán, quá trình mã hóa sang PDDL/SAS+, đến cách planner tìm kiếm lời giải hoặc chứng minh vô nghiệm.

Mục tiêu chính của hệ thống bao gồm:

- Cung cấp giao diện đồ họa để nhập đồ thị $G = (V, E)$, tập start I_s và goal I_t thông qua thao tác click/drag-and-drop, không yêu cầu người dùng biết định dạng file.
- Chạy thực sự solver PARIS trên backend cho các instance nhỏ ($n \leq 50$) và hiển thị kết quả trực tiếp từ log thực thi.
- Trực quan hóa từng bước token jump trong chuỗi reconfiguration thông qua sequence player có animation.
- Giải thích thuật toán PARIS ở cấp độ khái niệm và chi tiết kỹ thuật, bao gồm mã hóa PDDL/SAS+, heuristic search với landmarks, symbolic search bằng BDD, và counter abstraction.
- Đảm bảo toàn bộ hệ thống hoạt động với chi phí vận hành gần bằng 0, ưu tiên nền tảng free-tier và mã nguồn mở.

2 Phân tích người dùng và use-case

2.1 Đối tượng người dùng

Hệ thống phục vụ ba nhóm đối tượng chính:

- **Sinh viên / nhà nghiên cứu mới:** Muốn hiểu mô hình hóa reconfiguration, tại sao ISR được giải bằng planning, và ý nghĩa các thống kê đầu ra.
- **Người dùng solver:** Có instance nhỏ và muốn hiểu vì sao solver kết luận reachable/unreachable, vì sao chạy lâu, cần xem log và chuỗi kết quả.
- **Giảng viên:** Cần công cụ minh họa trực quan cho bài giảng về combinatorial reconfiguration và planning-based methods.

2.2 Các use-case chính

1. **UC-01 – Nhập instance:** Người dùng tạo đồ thị qua UI, chọn I_s và I_t, hoặc import file `.col`/`.dat` chuẩn CoRe 2022.
2. **UC-02 – Chạy solver:** Gửi instance lên backend, backend gọi PARIS thật, trả về log và kết quả.
3. **UC-03 – Xem chuỗi kết quả:** Dùng sequence player xem từng bước token jump, hiểu tại sao bước đó hợp lệ.
4. **UC-04 – Học thuật toán:** Đọc trang giải thích algorithm: mã hóa PDDL, heuristic, BDD, counter abstraction.
5. **UC-05 – Phân tích vô nghiệm:** Với instance vô nghiệm, hiểu tại sao không có chuỗi hợp lệ (qua counter abstraction).

3 Thiết kế kiến trúc chi tiết

Hệ thống được thiết kế theo mô hình ba tầng tách biệt hoàn toàn:

Hình 1: Kiến trúc ba tầng của hệ thống

3.1 Frontend (React + TypeScript)

Frontend chịu trách nhiệm toàn bộ giao diện người dùng:

- **Graph Editor:** Dùng thư viện Cytoscape.js để hiển thị đồ thị, hỗ trợ click để thêm/xóa đỉnh và cạnh, drag-and-drop để di chuyển đỉnh, undo/redo cơ bản.
- **Token Selector:** Click lên đỉnh để toggle token, hệ thống tự kiểm tra tính hợp lệ của independent set và hiển thị cảnh báo nếu vi phạm adjacency constraint.
- **Sequence Player:** Thanh timeline từ bước 0 đến bước L, nút play/pause/next/prev, mỗi bước highlight token bị di chuyển và đỉnh đích.
- **Algorithm Explainer:** Trang giải thích tĩnh kết hợp các diagram TikZ-style được render bằng SVG.
- **Log Viewer:** Hiển thị log thực thi thô từ PARIS, parse và làm nổi bật các thông tin quan trọng.

3.2 Backend (Python / FastAPI)

Backend xử lý logic nghiệp vụ và giao tiếp với solver:

- **Input Validation:** Kiểm tra đồ thị hợp lệ, kiểm tra I_s và I_t là independent set, kiểm tra $|I_s| = |I_t|$, giới hạn $n \leq 50$ cho chế độ online.
- **File Generation:** Tự động sinh file `.col` (định dạng DIMACS) và file `.dat` (start/-goal) theo chuẩn CoRe 2022.
- **Solver Invocation:** Gọi PARIS qua `subprocess` với timeout cứng (mặc định 60 giây), bắt `stdout`/`stderr` để lấy log.
- **Output Parsing:** Parse file `.out` để lấy YES/NO và chuỗi trạng thái, chuyển sang JSON.
- **Cache:** Lưu kết quả theo hash của $(G, I_s, I_t, \text{option})$ để tránh chạy lại instance đã có.

3.3 Solver Layer (PARIS + Docker)

- PARIS được đóng gói trong Docker image để đảm bảo môi trường thực thi nhất quán.
- Backend gọi Docker container qua `docker run` hoặc trực tiếp gọi binary nếu môi trường cho phép.
- Giới hạn tài nguyên: `-memory=512m`, `-cpus=1.0`, timeout 60 giây.

4 Thiết kế giao diện người dùng (UI/UX)

Giao diện được chia thành 4 khu vực chức năng chính:

#	Khu vực	Nội dung
1	Input Panel (trái)	Nhập số đỉnh, bộ ví dụ sẵn có, import/export file .col/.dat
2	Graph Canvas (giữa)	Cytoscape.js canvas, hiển thị đồ thị và token, thao tác drag-and-drop
3	Control Panel (phải)	Nút "Run Solver", chọn mode, hiển thị status và log
4	Sequence Player (dưới)	Thanh timeline, nút điều khiển, mô tả bước hiện tại

Bảng 1: Bố cục giao diện người dùng

5 Mô hình dữ liệu và luồng xử lý

5.1 Cấu trúc dữ liệu chính

Listing 1: Cấu trúc dữ liệu cốt lõi (Python/Pydantic)

```python
class ISRInstance(BaseModel):
    vertices: List[int]                 # Danh sach dinh: [1, 2, ..., n]
    edges: List[Tuple[int, int]]        # Danh sach canh: [(u1,v1), ...]
    start: List[int]                    # I_s: tap dinh co token ban dau
    goal: List[int]                     # I_t: tap dinh co token muc tieu

class SolverResult(BaseModel):
    status: Literal["YES", "NO", "TIMEOUT", "ERROR"]
    sequence: Optional[List[List[int]]]  # Chuoi reconfiguration neu co
    length: Optional[int]                # Do dai chuoi
    runtime_seconds: float
    log: str                             # Log tho cua PARIS
    cached: bool
```

5.2 Luồng xử lý chi tiết

1. Người dùng nhập đồ thị và chọn I_s, I_t qua UI.
2. Frontend gửi `POST /api/solve` với payload `ISRInstance`.
3. Backend kiểm tra tính hợp lệ: độc lập, kích thước bằng nhau, $n \leq 50$.
4. Backend tính hash input, kiểm tra cache. Nếu đã có, trả về ngay.
5. Nếu chưa có: sinh file `graph.col` và `instance.dat`.
6. Gọi PARIS: `./paris graph.col instance.dat -timeout 60`.
7. Bắt stdout/stderr, parse file `output.out`.
8. Lưu kết quả vào cache, trả về `SolverResult` dưới dạng JSON.
9. Frontend render sequence player nếu `status == "YES"`.

6 Kế hoạch phát triển theo sprint (10 tuần)

Sprint	Nội dung và deliverable
1	Đọc kỹ bài báo PARIS (Christen et al., ECAI 2023), nghiên cứu định dạng file CoRe 2022, cài đặt và chạy thử PARIS từ Zenodo. Deliverable: PARIS chạy được local, hiểu rõ input/output.
2	Thiết kế kiến trúc hệ thống, chọn công nghệ, thiết lập GitHub repo, CI/CD cơ bản, khung React app và FastAPI app. Deliverable: repo với skeleton code, workflow deploy GitHub Pages.
3	Backend: API endpoint `/api/solve`, validate input, sinh file `.col`/`.dat`, gọi PARIS qua subprocess, parse output `.out`. Deliverable: backend gọi được PARIS và trả về JSON.
4	Tích hợp Docker cho PARIS, timeout/quota, cache theo hash. Test với ít nhất 3 instance. Deliverable: backend ổn định, có cache và giới hạn tài nguyên.
5	Frontend: graph editor với Cytoscape.js, chọn token start/goal, overlay constraint, kiểm tra hợp lệ và thông báo lỗi. Deliverable: graph editor hoạt động.
6	Frontend: kết nối với backend API, hiển thị log thực thi, hiển thị YES/NO và chuỗi kết quả thô. Bộ ví dụ tối thiểu 8 instance. Deliverable: end-to-end flow hoạt động.
7	Sequence player hoàn chỉnh: thanh timeline, play/pause/next/prev, animation token jump, highlight token di chuyển và đỉnh đích. Deliverable: sequence player đầy đủ.
8	Trang giải thích thuật toán: mô hình trạng thái ISR, encoding PDDL/SAS+, heuristic search, symbolic search (BDD), counter abstraction, portfolio. Deliverable: trang algorithm explain hoàn chỉnh.
9	Testing toàn hệ thống, tối ưu hiệu năng, sửa bug, deploy backend lên Render/Railway free-tier, verify deploy. Deliverable: hệ thống deploy được.
10	Hoàn thiện tài liệu (README, báo cáo kỹ thuật, slide, video demo), kiểm tra toàn bộ checklist nghiệm thu cuối kỳ, nộp bài. Deliverable: toàn bộ deliverables hoàn chỉnh.

Bảng 2: Kế hoạch phát triển theo sprint

7 Phân công nhiệm vụ chi tiết

Thành viên	**Vai trò**	**Nhiệm vụ chi tiết**
Phương, Uy	Backend / Solver Integration	Xây dựng FastAPI server, tích hợp PARIS qua Docker/subprocess, sinh và parse file `.col`/`.dat`/`.out`, cache, validate input, giới hạn timeout/quota, deploy backend.
Sơn, Đức	Frontend / Visualization	Xây dựng React app, graph editor (Cytoscape.js), token selector, sequence player, constraint overlay, connect với backend API, deploy GitHub Pages, GitHub Actions.
An, Quang	Research & Documentation	Nghiên cứu thuật toán PARIS (bài báo ECAI 2023), viết Algorithm Mechanism Report, trang giải thích thuật toán trong webapp, chuẩn bị bộ ≥ 8 instance ví dụ, viết báo cáo kỹ thuật, slide, video demo.

Bảng 3: Phân công nhiệm vụ theo vai trò

8 Công nghệ sử dụng

Thành phần	Công nghệ	Lý do lựa chọn
Frontend framework	React + TypeScript	Quản lý state phức tạp tốt, build tĩnh cho GitHub Pages.
Graph visualization	Cytoscape.js	Hỗ trợ drag-and-drop, layout tự động, highlight, phù hợp bài toán graph.
Backend framework	Python / FastAPI	Tích hợp tốt với subprocess để gọi PARIS, async support, validation dễ với Pydantic.
Solver container	Docker + PARIS	Đảm bảo môi trường nhất quán, giới hạn tài nguyên dễ dàng.
Frontend deploy	GitHub Pages + Actions	Miễn phí, tự động deploy khi push.
Backend deploy	Render / Railway (free-tier)	Hỗ trợ Docker container, free-tier đủ cho instance nhỏ.
Version control	Git / GitHub	Quản lý code, issue tracking, CI/CD tích hợp.

Bảng 4: Bảng công nghệ sử dụng

9 Triển khai và tối ưu chi phí

Toàn bộ hệ thống được triển khai trên nền tảng free-tier:

- **Frontend:** GitHub Pages – hoàn toàn miễn phí.
- **Backend:** Render hoặc Railway (free-tier) – hỗ trợ Docker, đủ cho instance nhỏ.
- **Chi phí ước tính:** \$0/tháng nếu lưu lượng thấp.

Cơ chế kiểm soát chi phí:

- Giới hạn cứng kích thước instance: $n \leq 50$ đỉnh cho chế độ online.
- Timeout 60 giây mỗi request.
- Cache kết quả theo hash input.
- Rate limiting: tối đa 5 request mỗi phút mỗi IP.

10 Quản lý rủi ro

Rủi ro	**Mức độ**	**Biện pháp giảm thiểu**
Khó tích hợp PARIS binary vào backend	Cao	Đọc kỹ README Zenodo tuần 1; thử nghiệm sớm (sprint 3); dùng Docker để tránh vấn đề môi trường.
Backend free-tier hết RAM khi chạy PARIS	Trung bình	Giới hạn $n \leq 50$; đặt timeout cứng; cung cấp chế độ offline nếu cần.
Cytoscape.js khó tùy biến animation	Thấp	Dùng built-in style API; fallback là highlight đơn giản không dùng animation.
Thiếu thời gian hoàn thiện trang giải thích	Trung bình	Ưu tiên nội dung text+diagram tĩnh trước; animation là tính năng mở rộng.

Bảng 5: Bảng quản lý rủi ro

Tài liệu

[1] Remo Christen et al. *PARIS: Planning Algorithms for Reconfiguring Independent Sets*. ECAI 2023, pp. 453–460. https://ai.dmi.unibas.ch/papers/christen-et-al-ecai2023.pdf.

[2] Remo Christen and collaborators. *Code and data for PARIS* (Zenodo). https://zenodo.org/records/8178793, 2023.

[3] CoRe Challenge 2022 Organizers. *CoRe Challenge 2022 website*. https://core-challenge.github.io/2022/.