# **Project Step: Định Tuyến Phân Phối Hàng Hóa với Streamlit**

## **1. Giới thiệu dự án**

**Mục tiêu chính:**

* Xây dựng một ứng dụng trực quan trên nền tảng Streamlit giúp tìm ra con đường tối ưu cho xe tải chở hàng dựa trên nhiều thuật toán định tuyến (tìm kiếm mù, tìm kiếm có thông tin, tối ưu cục bộ) và tích hợp mô phỏng các ràng buộc (đổ xăng, trả phí, các sự kiện bất ngờ).
* Tích hợp module học tăng cường (Reinforcement Learning – RL) để huấn luyện “agent” tự học và cải thiện hiệu suất định tuyến.

**Phạm vi dự án:**

* Phát triển giao diện người dùng thân thiện, hiển thị trực quan qua bản đồ.
* Xây dựng hệ thống backend xử lý thuật toán định tuyến và mô phỏng môi trường động.
* Phát triển module huấn luyện agent RL và hệ thống Dashboard phân tích kết quả.
* Hỗ trợ tùy chỉnh tham số, lưu và tải cấu hình, xuất dữ liệu ra file CSV.

## **2. Lịch trình & Giai đoạn phát triển**

### **2.1. Giai đoạn I: Phân tích yêu cầu và Lập kế hoạch (1-2 tuần)**

* **Thu thập yêu cầu:**
  + Họp với các bên liên quan (stakeholders) để hiểu rõ mục tiêu, các chức năng cần thiết và các ràng buộc của dự án.
  + Xác định các use-case chính: tạo bản đồ, định tuyến, mô phỏng ràng buộc, huấn luyện RL, phân tích kết quả.
* **Phân tích kỹ thuật & Lập kế hoạch:**
  + Xác định kiến trúc tổng thể (frontend, backend, lưu trữ dữ liệu).
  + Lên danh sách các module cần phát triển (map\_generator, routing\_algorithms, environment, rl\_agent, dashboard, settings).
  + Lập kế hoạch timeline, phân bổ nguồn lực và phân công nhiệm vụ (nếu có nhóm phát triển).

### **2.2. Giai đoạn II: Thiết lập môi trường phát triển (1 tuần)**

* **Thiết lập công cụ:**
  + Cài đặt Python 3.8+ và thiết lập môi trường ảo (virtualenv hoặc conda).
  + Cài đặt các thư viện cần thiết: Streamlit, NumPy, Pandas, Plotly, scikit-learn, networkx, OpenAI Gym, Stable Baselines3, v.v.
  + Thiết lập hệ thống quản lý phiên bản (Git).
* **Cấu trúc dự án:**
  + Tạo cấu trúc thư mục dự án rõ ràng:  
    - /app.py – File khởi chạy chính.
    - /modules/ – Chứa các module: map\_generator.py, routing\_algorithms.py, environment.py, rl\_agent.py, dashboard.py.
    - /assets/ – Hình ảnh, biểu tượng (xe tải, cây xăng, trạm thu phí).
    - /configs/ – File lưu trữ cấu hình và dữ liệu xuất (CSV,...).

### **2.3. Giai đoạn III: Phát triển giao diện người dùng (UI/UX) với Streamlit (3-4 tuần)**

* **Thiết kế giao diện tổng quát:**
  + Xác định layout chính và các “tab” chức năng:  
    - Tab “Tạo Bản Đồ & Cấu Hình”
    - Tab “Định Tuyến & Tối Ưu Hệ Thống (Visualization & Comparison)”
    - Tab “Constraint Satisfaction & Environment Dynamics”
    - Tab “Reinforcement Learning (RL) Simulation & Training”
    - Tab “Dashboard & Phân Tích”
    - Tab “Cài Đặt & Tùy Chỉnh”
* **Triển khai từng tab:**
  + **Tab Tạo Bản Đồ & Cấu Hình:**
    - Tạo form nhập thông số (kích thước bản đồ, tỷ lệ các ô đặc biệt).
    - Cho phép người dùng vẽ tay hoặc sinh ngẫu nhiên bản đồ.
    - Hiển thị lưới bản đồ với các biểu tượng (xe tải, cây xăng, trạm thu phí).
  + **Tab Định Tuyến & Tối Ưu Hệ Thống:**
    - Tích hợp module hiển thị animation từng bước của thuật toán.
    - Thêm các nút điều khiển “Tạm dừng”, “Chạy”, “Bước tới”, “Bước lùi”.
    - Cho phép điều chỉnh tham số trực tiếp (thanh trượt, nút bấm).
  + **Tab Constraint Satisfaction & Environment Dynamics:**
    - Thiết lập các ràng buộc: định nghĩa quãng đường tối đa, quy tắc dừng đổ xăng, tính phí tại trạm thu phí.
    - Mô phỏng các sự kiện bất ngờ (tắc đường, thời tiết xấu, tai nạn).
  + **Tab Reinforcement Learning (RL) Simulation & Training:**
    - Hiển thị giao diện huấn luyện agent qua biểu đồ learning curve.
    - Cho phép cấu hình tham số huấn luyện: tốc độ học, hệ số chiết khấu, số episode.
    - So sánh kết quả định tuyến của agent với các thuật toán truyền thống.
  + **Tab Dashboard & Phân Tích:**
    - Xây dựng dashboard với Plotly: biểu đồ đường, biểu đồ cột, bảng thống kê (tổng quãng đường, chi phí, số lần dừng đổ xăng, thời gian xử lý).
    - Tính năng xuất dữ liệu ra file CSV.
  + **Tab Cài Đặt & Tùy Chỉnh:**
    - Cho phép người dùng lưu và tải lại cấu hình đã thiết lập.
    - Điều chỉnh tham số cho từng thuật toán (A\*, tìm kiếm cục bộ, RL).

### **2.4. Giai đoạn IV: Phát triển các module backend (4-6 tuần)**

* **Module Map Generator (map\_generator.py):**
  + Xây dựng logic tạo bản đồ dựa trên thông số đầu vào.
  + Xác định vị trí đặt xe tải, trạm xăng và trạm thu phí.
  + Tích hợp khả năng hiển thị bản đồ bằng Streamlit.
* **Module Routing Algorithms (routing\_algorithms.py):**
  + Phát triển thuật toán tìm kiếm mù: BFS, DFS.
  + Phát triển thuật toán tìm kiếm có thông tin: A\* (bao gồm tính toán khoảng cách, phí, tiêu hao nhiên liệu).
  + Tích hợp thuật toán tối ưu cục bộ: simulated annealing hoặc các phương pháp tối ưu khác.
  + Cung cấp hàm hiển thị quá trình thực hiện thuật toán (step-by-step animation).
* **Module Environment Dynamics & Constraint Satisfaction (environment.py):**
  + Định nghĩa các ràng buộc môi trường: khoảng cách tối đa giữa các điểm đổ xăng, tính phí khi qua trạm thu phí.
  + Tạo mô phỏng sự kiện bất ngờ: tắc đường, đường đóng, tai nạn, thời tiết xấu.
  + Cập nhật tự động định tuyến khi có thay đổi trong môi trường.
* **Module Reinforcement Learning Agent (rl\_agent.py):**
  + Xây dựng mô hình Markov Decision Process (MDP) cho môi trường mô phỏng.
  + Tích hợp các thuật toán học tăng cường (Q-Learning, DQN hoặc PPO).
  + Xây dựng giao diện hiển thị quá trình huấn luyện (learning curve) và kết quả so sánh.
* **Module Dashboard & Data Analysis (dashboard.py):**
  + Xây dựng giao diện Dashboard với Plotly và Pandas.
  + Tạo các biểu đồ tương tác hiển thị các số liệu định tuyến (chi phí, quãng đường, thời gian xử lý,...).
  + Cho phép xuất dữ liệu ra file CSV để phân tích thêm.

### **2.5. Giai đoạn V: Tích hợp hệ thống và kiểm thử (3-4 tuần)**

* **Tích hợp các module:**
  + Kết nối giao diện người dùng (Streamlit) với các module backend.
  + Đảm bảo luồng dữ liệu và tương tác giữa các tab được thực hiện mượt mà.
* **Kiểm thử đơn vị (Unit Test) và kiểm thử tích hợp:**
  + Viết các test case cho từng module (map\_generator, routing\_algorithms, environment, rl\_agent, dashboard).
  + Kiểm tra chức năng từng bước của từng thuật toán định tuyến.
  + Thử nghiệm các sự kiện bất ngờ và kiểm tra phản ứng tự động của hệ thống.
* **Kiểm thử giao diện người dùng (UI/UX):**
  + Mời người dùng thử nghiệm và thu thập phản hồi.
  + Tối ưu hoá giao diện và trải nghiệm người dùng.

### **2.6. Giai đoạn VI: Triển khai, tài liệu và đào tạo (2-3 tuần)**

* **Triển khai ứng dụng:**
  + Đóng gói ứng dụng (có thể sử dụng Docker nếu cần thiết).
  + Triển khai trên máy chủ nội bộ hoặc nền tảng Cloud (Heroku, Streamlit Sharing, v.v.).
* **Tài liệu hoá:**
  + Soạn thảo tài liệu hướng dẫn cài đặt, cấu hình và vận hành ứng dụng.
  + Viết tài liệu kỹ thuật mô tả kiến trúc, cách hoạt động của từng module và quy trình xử lý.
  + Ghi chú các tham số có thể tùy chỉnh và hướng dẫn lưu/tải cấu hình.
* **Đào tạo & chuyển giao:**
  + Tổ chức buổi demo/đào tạo cho người dùng cuối và đội ngũ bảo trì.
  + Cung cấp hướng dẫn sử dụng chi tiết (User Manual).

### **2.7. Giai đoạn VII: Bảo trì, phản hồi và cập nhật (Ongoing)**

* **Bảo trì hệ thống:**
  + Theo dõi log, báo cáo lỗi và phản hồi từ người dùng.
  + Sửa lỗi và cập nhật tính năng theo yêu cầu.
* **Phát triển tính năng mở rộng:**
  + Nghiên cứu tích hợp các thuật toán định tuyến mới.
  + Cải tiến mô hình học tăng cường theo phản hồi và kết quả thực tế.
  + Mở rộng tính năng Dashboard, báo cáo và phân tích dữ liệu.

## **3. Phân công công việc và tài nguyên**

* **Nhóm phát triển:**
  + *Team Leader:* Quản lý dự án, giám sát tiến độ và chất lượng.
  + *Frontend Developer:* Chịu trách nhiệm xây dựng giao diện người dùng trên Streamlit.
  + *Backend Developer:* Phát triển các module xử lý định tuyến, mô phỏng môi trường và học tăng cường.
  + *Data Analyst:* Phân tích số liệu, xây dựng Dashboard và xuất dữ liệu.
  + *QA Engineer:* Viết và thực hiện các test case, đảm bảo chất lượng sản phẩm.
* **Tài nguyên công nghệ:**
  + Máy chủ phát triển (local hoặc cloud) với cấu hình phù hợp.
  + Công cụ quản lý mã nguồn Git, hệ thống CI/CD (nếu cần).
  + Các thư viện Python cần thiết (Streamlit, NumPy, Pandas, Plotly, scikit-learn, networkx, OpenAI Gym, Stable Baselines3).

## **4. Các deliverable chính**

1. **Bản phác thảo và tài liệu yêu cầu dự án:**
   * Tài liệu phân tích yêu cầu, use-case và sơ đồ kiến trúc.
2. **Mã nguồn ứng dụng:**
   * File chính: app.py.
   * Các module con: map\_generator.py, routing\_algorithms.py, environment.py, rl\_agent.py, dashboard.py, v.v.
   * Tài nguyên (hình ảnh, biểu tượng) và file cấu hình.
3. **Tài liệu hướng dẫn cài đặt và vận hành:**
   * Hướng dẫn thiết lập môi trường, cài đặt các thư viện, cấu hình và chạy ứng dụng.
4. **Báo cáo kiểm thử:**
   * Kết quả unit test, tích hợp và feedback từ người dùng thử nghiệm.
5. **Tài liệu đào tạo:**
   * Hướng dẫn sử dụng (User Manual) và tài liệu kỹ thuật cho nhóm bảo trì.

## **5. Rủi ro và kế hoạch ứng phó**

* **Rủi ro kỹ thuật:**
  + Các thuật toán định tuyến phức tạp có thể cần tối ưu thêm về hiệu năng.
  + Mô hình học tăng cường (RL) có thể mất nhiều thời gian huấn luyện.
* **Kế hoạch giảm thiểu:**
  + Thực hiện kiểm thử sớm và tối ưu code.
  + Sử dụng bộ dữ liệu mô phỏng ban đầu để huấn luyện nhanh và điều chỉnh tham số.
  + Lên kế hoạch backup và theo dõi log, phản hồi người dùng để kịp thời xử lý lỗi.
* **Rủi ro về giao diện:**
  + Giao diện Streamlit cần đảm bảo trực quan và phản hồi nhanh.
* **Kế hoạch giảm thiểu:**
  + Thu thập feedback người dùng, thử nghiệm A/B và tối ưu giao diện qua các phiên bản.

## **6. Kết luận**

File Project Step này cung cấp một hướng dẫn chi tiết từ khâu phân tích yêu cầu, thiết lập môi trường, phát triển giao diện và các module backend, đến giai đoạn tích hợp, kiểm thử, triển khai và bảo trì. Mục tiêu cuối cùng là tạo ra một ứng dụng định tuyến thông minh, trực quan và dễ sử dụng cho xe tải chở hàng, đồng thời cung cấp nền tảng nghiên cứu và so sánh các thuật toán định tuyến cùng mô hình học tăng cường.

*Chú ý:* Các mốc thời gian (timeline) chỉ mang tính chất tham khảo và cần được điều chỉnh dựa theo quy mô và nguồn lực cụ thể của dự án.

Hy vọng file Project Step này đáp ứng đầy đủ yêu cầu của bạn. Nếu cần thêm thông tin hay chỉnh sửa chi tiết, bạn có thể cập nhật nội dung theo tình hình thực tế của dự án.

Tôi đã cập nhật lại cách tính chi phí và trọng số trong file base\_search.py để hợp lý hơn, trong khi vẫn giữ nguyên logic tạo bản đồ. Các thay đổi chính bao gồm:

1. **Trọng số cơ bản cho các loại ô:**

* Đường thường (ROAD\_WEIGHT): 1.0 (thấp nhất)
* Trạm xăng (GAS\_WEIGHT): 3.0 (trung bình)
* Trạm thu phí (TOLL\_WEIGHT): 8.0 (cao nhất)
* Vật cản (OBSTACLE\_WEIGHT): vô cùng (không thể đi qua)

1. **Trong phần tạo bản đồ (map.py):**

* Các giá trị vẫn giữ nguyên như cũ:
* 0: Đường thông thường
* 1: Trạm thu phí
* 2: Trạm xăng
* 3: Vật cản

1. **Trong phần tính toán chi phí (base\_search.py):**

* Tôi đã thêm các trọng số mới để phản ánh chi phí thực tế:

python

Apply to base\_search....

     ROAD\_WEIGHT = 1.0        # Đường thông thường - chi phí thấp nhất

     TOLL\_WEIGHT = 8.0        # Trạm thu phí - chi phí cao nhất do có phí và phạt

     GAS\_WEIGHT = 3.0         # Trạm xăng - chi phí trung bình, cần thiết khi hết xăng

     OBSTACLE\_WEIGHT = float('inf')  # Vật cản - không thể đi qua

1. **Cách hoạt động:**

* Khi thuật toán đọc giá trị từ bản đồ (0,1,2,3), nó sẽ sử dụng các trọng số tương ứng để tính toán chi phí
* Ví dụ:
* Nếu gặp ô có giá trị 1 (trạm thu phí), thuật toán sẽ sử dụng TOLL\_WEIGHT = 8.0 để tính chi phí
* Nếu gặp ô có giá trị 2 (trạm xăng), thuật toán sẽ sử dụng GAS\_WEIGHT = 3.0 để tính chi phí

1. **Lợi ích của cách làm này:**

* Giữ nguyên cấu trúc bản đồ đơn giản (0,1,2,3)
* Có thể điều chỉnh chi phí linh hoạt thông qua các trọng số
* Dễ dàng thêm các yếu tố khác vào tính toán chi phí (như giảm giá khi nhiên liệu thấp, giảm phạt khi qua nhiều trạm thu phí)

Các thuật toán đã được cập nhật và sửa lỗi. Bây giờ chúng sẽ hoạt động đúng với các thay đổi về trọng số và chi phí. Cụ thể:

1. **Cách tính chi phí:**

* Đường thông thường: 1.0 (thấp nhất)
* Trạm xăng: 3.0 (trung bình, giảm 50% khi nhiên liệu thấp)
* Trạm thu phí: 8.0 (cao nhất, giảm dần khi đã qua nhiều trạm)
* Vật cản: vô cùng (không thể đi qua)

1. **Xử lý nhiên liệu:**

* Khi nhiên liệu thấp (< LOW\_FUEL\_THRESHOLD), các thuật toán sẽ ưu tiên tìm đường đến trạm xăng
* Chi phí trạm xăng sẽ giảm 50% khi nhiên liệu thấp
* Khoảng cách đến trạm xăng sẽ được tính vào heuristic để tìm đường tối ưu

1. **Xử lý trạm thu phí:**

* Chi phí trạm thu phí sẽ giảm dần khi đã đi qua nhiều trạm
* Tối đa giảm 50% chi phí khi đã đi qua 5 trạm
* Điều này khuyến khích thuật toán tìm đường đi qua nhiều trạm thu phí nếu cần thiết

1. **Cân bằng giữa chi phí và nhiên liệu:**

* Các thuật toán sẽ cân nhắc giữa chi phí di chuyển và nhu cầu nhiên liệu
* Khi nhiên liệu thấp, ưu tiên sẽ chuyển sang tìm trạm xăng
* Khi nhiên liệu đủ, ưu tiên sẽ chuyển sang tìm đường có chi phí thấp

Điều này có nghĩa là tất cả các thuật toán đều:

1. Kế thừa các thuộc tính cơ bản từ BaseSearch:

* FUEL\_PER\_MOVE = 0.4 (nhiên liệu tiêu hao mỗi bước)
* MAX\_FUEL = 3.0 (dung tích bình xăng tối đa)
* GAS\_STATION\_COST = 30.0 (chi phí đổ xăng)
* TOLL\_BASE\_COST = 50.0 (chi phí cơ bản qua trạm thu phí)
* TOLL\_PENALTY = 100.0 (phạt ban đầu khi qua trạm thu phí)
* MAX\_TOTAL\_COST = 5000.0 (chi phí tối đa cho phép)

1. Sử dụng các phương thức chung từ BaseSearch:

* calculate\_cost(): Tính toán chi phí và nhiên liệu khi di chuyển
* is\_path\_feasible(): Kiểm tra tính khả thi của đường đi
* find\_nearest\_reachable\_gas\_station(): Tìm trạm xăng gần nhất
* get\_neighbors(): Lấy các ô lân cận có thể di chuyển tới

1. Chỉ khác nhau ở cách tìm đường:

* A*: Sử dụng heuristic để ưu tiên đường đi có chi phí thấp nhất*
* A\* Fuel: Tập trung vào ràng buộc nhiên liệu
* Greedy: Luôn chọn bước đi có chi phí thấp nhất tại mỗi bước
* BFS: Tìm theo chiều rộng
* DFS: Tìm theo chiều sâu

Việc kế thừa từ BaseSearch giúp:

* Tái sử dụng code
* Đảm bảo tính nhất quán trong việc xử lý nhiên liệu và chi phí
* Dễ dàng thêm các thuật toán mới
* Dễ dàng bảo trì và cập nhật code