NATIONAL ECONOMICS UNIVERSITY

Hanoi College of Technology

INTRODUCTION TO AI (TOKT11121.AI66A)

REPORT

Instructor: Nguyễn Trọng Nghĩa

Student: Trần Tuấn Anh - 11247264

Hoàng Thị Ngọc Hân - 11247286

Nguyễn Phùng Hóa - 11247290

Nguyễn Hoàng Tuấn - 11247363

Hà Nội, ngày 17 tháng 10 năm 2025

Contents

1 Introduction

Trong thời đại công nghệ số và thương mại điện tử phát triển mạnh mẽ, các công ty giao hàng (delivery, logistics) đối mặt với một bài toán cơ bản nhưng vô cùng quan trọng: làm thế nào để tối ưu hóa lộ trình vận chuyển hàng hóa từ kho đến các điểm giao sao cho tổng chi phí thấp nhất và thời gian nhanh nhất. Đây không chỉ là vấn đề về giao thông hay bản đồ, mà là một bài toán tối ưu trong trí tuệ nhân tạo (AI) — thuộc nhóm tìm kiếm có thông tin (Informed Search).

Bài toán này có thể được mô hình hóa bằng đồ thị (graph), trong đó:

- Đỉnh (node) đại diện cho các địa điểm giao hàng (kho, trung tâm, khách hàng);
- Cạnh (edge) thể hiện quãng đường hoặc chi phí di chuyển giữa hai địa điểm;
- Trọng số (weight) của cạnh thể hiện độ dài quãng đường, chi phí nhiên liệu hoặc thời gian giao hàng.

Mục tiêu là tìm đường đi ngắn nhất từ điểm khởi đầu (I) đến điểm đích (A). Trong báo cáo này, chúng ta mô phỏng một đồ thị với các nút B, C, D, E, F, G, K, M làm các điểm trung gian mà xe có thể ghé qua.

2 Method & Experiment

2.1 Problem definition

Công ty giao hàng cần tìm tuyến đường tối ưu từ điểm xuất phát I (kho hàng) đến điểm đích A (khách hàng cuối cùng). Mỗi cạnh có chi phí di chuyển $g(n)$ (vd: khoảng cách hay thời gian lái), và mỗi nút có một giá trị heuristic $h(n)$ (vd: thời gian dừng tại điểm đó hoặc ước lượng khoảng cách đến đích).

Mục tiêu: Minimize tổng chi phí $\sum g(n)$ dọc theo đường đi từ I đến A.

2.2 Algorithms

Trong báo cáo này ta triển khai và so sánh ba thuật toán:

1. BFS (Breadth-First Search)
2. GBFS (Greedy Best-First Search)
3. A* (A-star)

2.2.1 BFS (Breadth-First Search)

BFS là thuật toán duyệt theo chiều rộng. Nếu các cạnh có trọng số bằng nhau (hoặc ta chỉ cần số bước tối thiểu), BFS sẽ tìm đường đi có số bước ít nhất.

Ưu điểm:

- Dễ cài đặt.
- Tìm được đường đi ngắn nhất trong đồ thị vô trọng số.

Nhược điểm:

- Không phù hợp với đồ thị có trọng số khác nhau.
- Tốn bộ nhớ và thời gian khi đồ thị lớn do mở rộng theo tầng.

Algorithm 1 Breadth-First Search (BFS)

```
Require: start node s, goal node t
queue ← [s]                                   ▷ FIFO queue
visited ← {s}
parent[s] ← NIL
while queue not empty do
    v ← Dequeue(queue)
    if v = t then
        return ReconstructPath(parent, t)
    end if
    for each neighbor u of v do
        if u ∉ visited then
            add u to visited
            parent[u] ← v
            Enqueue(queue, u)
        end if
    end for
end while
return FAILURE
```

2.2.2 Greedy Best-First Search (GBFS)

GBFS chọn nút để mở rộng dựa trên giá trị heuristic $h(n)$ nhỏ nhất, nghĩa là luôn đi theo nút có vẻ gần đích nhất theo ước lượng.

Ưu điểm:

- Thường tìm đường nhanh (ít nút được mở rộng) nếu heuristic tốt.

Nhược điểm:

- Không đảm bảo tìm đường ngắn nhất (vì bỏ qua chi phí thực tế $g(n)$).
- Dễ mắc kẹt local minima nếu heuristic sai lệch.

Algorithm 2 Greedy Best-First Search (GBFS)

```
Require: start s, goal t, heuristic h(·)
open ← priorityqueueorderedbyh(n)
    Insert(open, s)
    parent[s] ← NIL
    visited ← ∅
    while open not empty do
        v ← ExtractMin(open)                          ▷ node with smallest h
        if v = t then
            return ReconstructPath(parent, t)
        end if
        if v ∉ visited then
            add v to visited
            for each neighbor u of v do
                if u ∉ visited then
                    parent[u] ← v
                    Insert(open, u)
                end if
            end for
        end if
    end while
    return FAILURE
```

2.2.3 A* (A-star)

A* sử dụng hàm đánh giá $f(n) = g(n) + h(n)$, trong đó $g(n)$ là chi phí thực tế từ start đến nút n, và $h(n)$ là heuristic ước lượng chi phí từ n đến goal. Nếu h là admissible (không đánh giá quá cao), A* sẽ tìm đường tối ưu.

Ưu điểm:

- Nếu heuristic thỏa điều kiện admissible và consistent, A* tìm được đường ngắn nhất.

- Thường mở rộng ít nút hơn BFS.

Nhược điểm:

- Tốn bộ nhớ (lưu open/closed lists).
- Hiệu suất phụ thuộc vào chất lượng heuristic.

Algorithm 3 A* Search

```
Require: start s, goal t, heuristic h(·)
open ← priority queue ordered by f(n) = g(n) + h(n)
g[s] ← 0
Insert(open, s with priority f(s) = h(s))
parent[s] ← NIL
closed ← ∅
while open not empty do
    v ← ExtractMin(open)
    if v = t then
        return ReconstructPath(parent, t)
    end if
    add v to closed
    for each neighbor u of v with edge cost w(v, u) do
        tentative_g ← g[v] + w(v, u)
        if u ∈ closed and tentative_g ≥ g[u] then
            continue
        end if
        if u ∉ open or tentative_g < g[u] then
            parent[u] ← v
            g[u] ← tentative_g
            f[u] ← g[u] + h(u)
            if u ∉ open then
                Insert(open, u with priority f[u])
            else
                DecreaseKey(open, u, f[u])
            end if
        end if
    end for
end while
return FAILURE
```

2.3 Pseudo-code notes

Các thuật toán trên đã được viết dưới dạng pseudo-code; nếu bạn có đoạn mã Python/Java/C++ cụ thể, mình có thể chuyển nguyên xi vào môi trường algorithm hoặc sử dụng lstlisting nếu muốn hiển thị code thực thi.

3 Evaluation & Description

Để đánh giá, ta có thể thực hiện các bước:

1. Thiết kế một đồ thị mẫu (ví dụ hình minh họa), gán trọng số cạnh và heuristic cho các nút.
2. Chạy từng thuật toán trên cùng một đồ thị, ghi lại: đường tìm được, tổng chi phí, số nút mở rộng, thời gian chạy.
3. So sánh về độ chính xác (đường tối ưu hay không), hiệu suất (thời gian, bộ nhớ), và tính ổn định khi heuristic thay đổi.

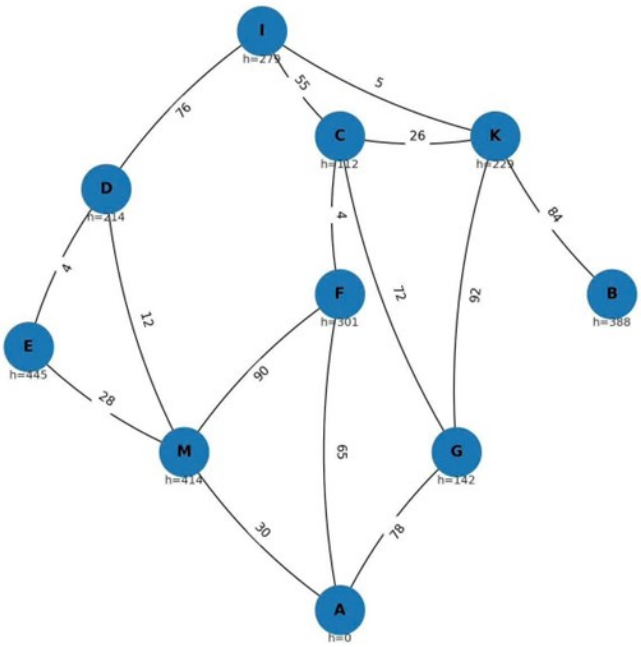

Figure 1: Ví dụ đồ thị giao hàng (placeholder).

4 Conclusion

Ba thuật toán BFS, GBFS và A* đều thuộc nhóm các thuật toán tìm kiếm trên đồ thị. BFS là cơ bản và phù hợp với đồ thị vô trọng số; GBFS nhanh

nếu heuristic tốt nhưng không đảm bảo tối ưu; A* là lựa chọn cân bằng khi có heuristic admissible — nó đảm bảo tìm đường tối ưu và thường hiệu quả hơn so với duyệt toàn bộ.

Trong thực tế, các hệ thống lớn (ví dụ tối ưu lộ trình cho nhiều xe, bản đồ động) thường kết hợp A* cho các đoạn cục bộ với các meta-heuristic (GA, Ant Colony) để tối ưu tổng lộ trình.

5 Discuss

5.1 Về heuristic

Hiệu quả của A* phụ thuộc rất lớn vào độ chính xác của hàm heuristic. Nếu h(n) được xác định bằng khoảng cách Euclid hoặc Manhattan trên bản đồ, A* hoạt động nhanh và chính xác. Nếu heuristic bị đánh giá quá cao, A* có thể trở nên không tối ưu; nếu quá thấp, nó sẽ hoạt động như Dijkstra và chậm hơn.

5.2 Mở rộng thực tế

Trong ứng dụng giao hàng quy mô lớn: Có thể áp dụng A* cho từng đoạn ngắn (micro-routing), sau đó dùng meta-heuristic (như Genetic Algorithm hoặc Ant Colony) để tối ưu tổng lộ trình.

Với bản đồ động (tắc đường, thay đổi chi phí), có thể dùng D* hoặc A*-Lite, cho phép cập nhật đường đi trong thời gian thực.

5.3 Tác động và ứng dụng

A* là nền tảng cho các hệ thống định tuyến hiện đại như Google Maps, Grab, UPS Route Optimization, robot tự hành.

Giúp tiết kiệm nhiên liệu, giảm thời gian vận chuyển, giảm chi phí logistics.

Trong AI, bài toán này còn mở rộng thành Path Planning trong robot, game, hoặc mô phỏng thành phố thông minh.

6 References

1. Albert Einstein. “On the electrodynamics of moving bodies”. Annalen der Physik 17 (1905), pp. 891–921.

2. Stuart J. Russell and Peter Norvig. Artificial Intelligence: A Modern Approach. 4th ed., Pearson, 2020. ISBN: 9780134610993.