Bài toán thu gom rác tối ưu

Nhóm 2

Nguyễn Ngân An - 11247253

Phạm Hữu Gia Ân - 11247254

Nguyễn Trọng Đại - 11247268

Lê Bá Phong - 11247339

Giảng viên hướng dẫn: TS. Nguyễn Trọng Nghĩa

Business AI Lab

1. Vấn đề trong thực tế
2. Công thức vấn đề
3. Thuật toán
4. Thử nghiệm, đánh giá, kết luận
5. Ứng dụng thực tế
6. Mở rộng
7. Thảo luận

1. Vấn đề trong thực tế

Business AI Lab

1. Vấn đề trong thực tế

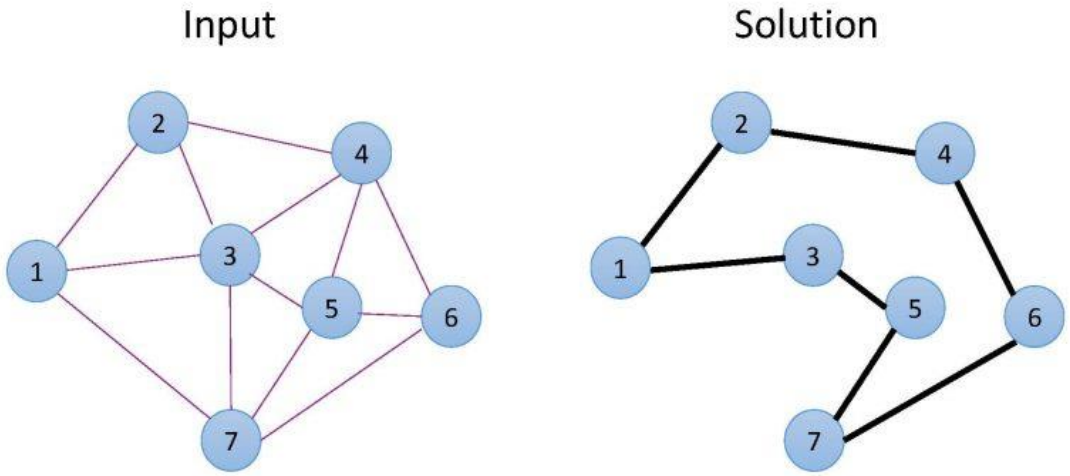
1.1. Bài toán thu gom rác

a. Mô tả vấn đề

Mỗi ngày xe rác cần đi qua hàng chục đến hàng trăm thùng rác rải rác trong khu phố và quay trở lại trạm với quãng đường ngắn nhất.

Tìm đường đi tối ưu giúp:

+ Tiết kiệm nhiên liệu
+ Giảm chi phí vận hành
+ Tăng hiệu suất thu gom, …



b. Dạng bài toán

- Mô hình hóa thành dạng bài toán:
  + Travelling Salesman Problem – TSP
  + Hamiltonian Path Problem – HPP

1. Vấn đề trong thực tế

1.2. Các tình huống thực tế khác

Shipper giao hàng

Khách du lịch đi thăm quan

Robot vận chuyển

2. Công thức vấn đề

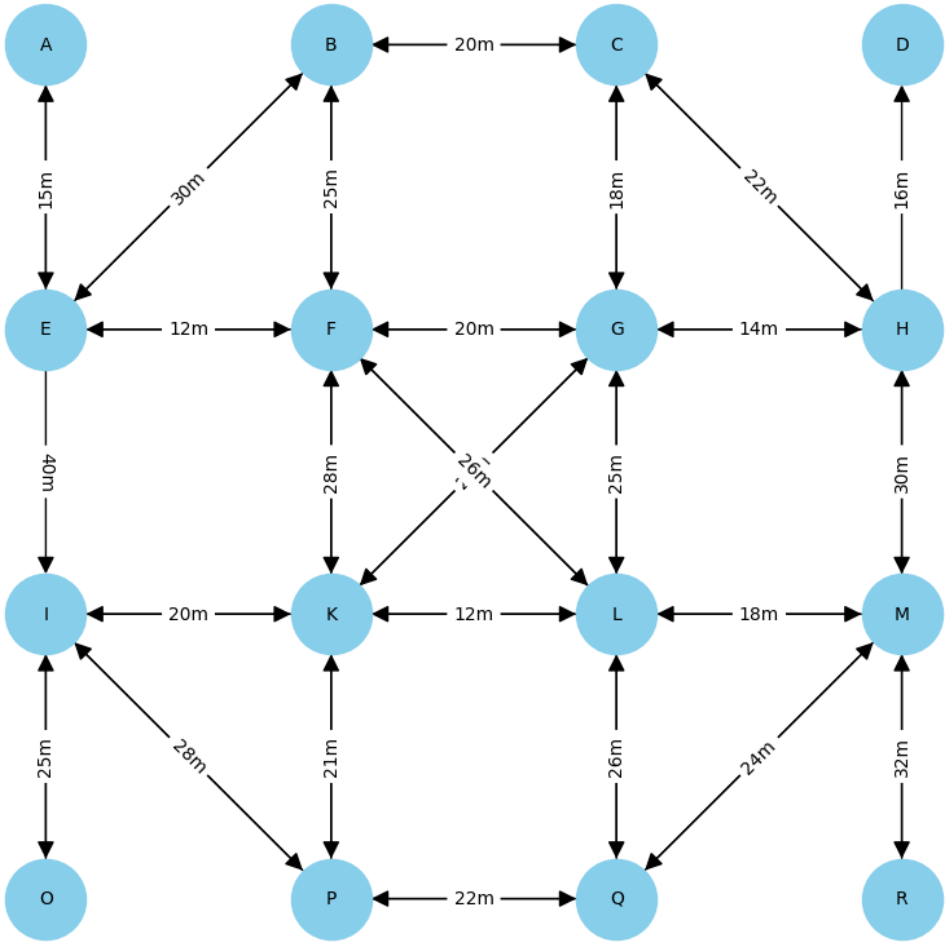
Business AI Lab

2. Công thức vấn đề

2.1. Mô phỏng bản đồ

Miêu tả:
- Mỗi node là một khu vực cần đi qua
- Cạnh nối giữa 2 node là đường đi xe thu gom rác

Một vài yếu tố khác:
- Đường một chiều: E -> I
- Đường cụt (có thể quay về): A, O, R
- Đường cụt (không thể quay về): D



2. Công thức vấn đề

2.2. Mô hình bài toán

Thành phần	Mô tả
Trạng thái	(Thành phố hiện tại, tập các thành phố đã đi qua)
Trạng thái bắt đầu	(A, {A})
Mục tiêu (Goal)	(A, {B, C, D, E, ...}) – đã thăm tất cả và quay lại A
Toán tử	Di chuyển đến một thành phố chưa thăm
Chi phí đường đi	Tổng khoảng cách đã di chuyển
Phỏng đoán (Heuristic)	Chi phí ước lượng tới các node chưa thăm

3. Thuật toán

1. Uniform Cost Search Algorithm
2. Greedy Best-First Sreach Algorithm
3. A* Algorithm

Business AI Lab

2025-10

3. Thuật toán

3.1. Uniform Cost Search Algorithm - UCS

Nguyên lý hoạt động	• Bắt đầu tại nút gốc và tiếp tục bằng cách duyệt các nút tiếp theo với trọng số hay chi phí thấp nhất tính từ nút gốc. • Thuật toán chọn để mở rộng nút có tổng chi phí từ gốc đến nút đó (g(n)) nhỏ nhất
Đặc điểm	• UCS không sử dụng heuristic. • Bảo đảm tìm được đường đi có chi phí nhỏ nhất nếu tất cả chi phí cạnh ≥ 0. • Tương đương với Dijkstra's Algorithm khi áp dụng cho đồ thị trọng số
Độ phức tạp	• Thời gian: O(n! * log(n!)) • Bộ nhớ: Tỷ lệ với số lượng nút đã mở rộng. (khá cao khi số node lớn)

3. Thuật toán

3.1. Uniform Cost Search Algorithm - UCS

Mã giả:

```
FUNCTION uniform_cost_search(graph, start, bins, TSP=False)
  bins ← set(bins); IF TSP: bins ← bins ∪ {start}
  pq ← min-heap [(0, start, frozenset(bins - {start}), [start])]
  visited ← {} ; expanded ← 0
  WHILE pq not empty:
    g, curr, remaining, path ← heapq.pop(pq); expanded += 1
    IF remaining empty:
      IF NOT TSP: RETURN (path, g, expanded)
      back_cost, back_path ← shortest_path(graph, curr, start)
      IF back_path: RETURN (path + back_path[1:], g + back_cost, expanded)
    IF (curr, remaining) in visited AND visited[(curr,remaining)] ≤ g: CONTINUE
    visited[(curr,remaining)] ← g
    FOR neighbor, info IN graph[curr].items():
      new_g ← g + info["distance"]
      new_rem ← frozenset(set(remaining) - {neighbor})
      heapq.push(pq, (new_g, neighbor, new_rem, path + [neighbor]))
  RETURN (None, ∞, expanded)
END
```

3. Thuật toán

3.1. Uniform Cost Search Algorithm - UCS

Ouput:

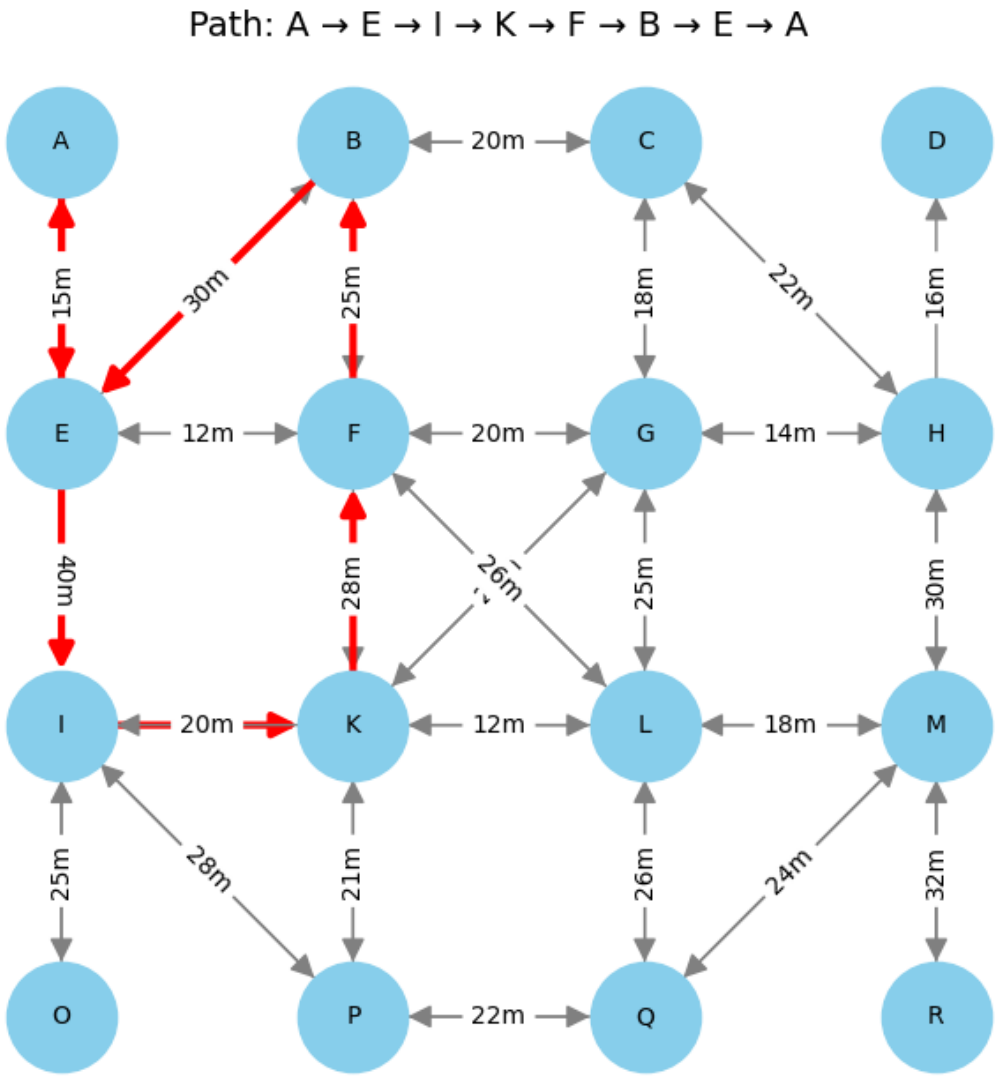
```
# ví dụ: start từ "A", thùng rác =  ["B", "I", "K"]
start = "A"
bins =  ["B", "I", "K"]

path, cost, expand_node = uniform_cost_search(graph, start, bins, TSP=True)
print("Đường đi (UCS):", path)
print("Tổng chi phí:", cost)
print("Số node mở rộng:", expand_node)

draw_graph(graph, path, size_g = 6)
```

✓ 0.6s

```
Đường đi (UCS): ['A', 'E', 'I', 'K', 'F', 'B', 'E', 'A']
Tổng chi phí: 173
Số node mở rộng: 229
```



3. Thuật toán

3.2. Greedy Best-First Search Algorithm

Nguyên lý hoạt động	• Thuật toán sẽ sử dụng 1 hàm đánh giá heuristic h(n) để đi từ nút hiện tại đến nút gần nhất và lặp lại cho đến khi đến nút cuối. • Hàm heuristic được sử dụng là Dijkstra: Tính chi phí ước lượng từ node hiện tại tới các bins và đi tới bins gần nhất, lặp lại cho đến khi đi hết và quay về.
Đặc điểm	• Tìm kiếm nhanh hơn UCS vì không quan tâm đến chi phí thực tế (g(n)). • Không đảm bảo tối ưu. • Phù hợp khi cần kết quả nhanh hoặc trong không gian tìm kiếm lớn.
Độ phức tạp	• Thời gian: $O(n^2 \log n)$ (Phụ thuộc mạnh vào chất lượng của heuristic.) • Bộ nhớ: Thấp

3. Thuật toán

3.2 Greedy Best-First Search Algorithm

Mã giả:

```
FUNCTION greedy_collect_bins(graph, start, bins, TSP=False)
  current ← start; path ← [start]; cost_total ← 0; remaining ← set(bins); expanded_total ← 0
  WHILE remaining:
    dist, prev, expanded ← dijkstra_greedy(graph, current); expanded_total += expanded
    next_bin ← argmin_{b∈remaining} dist[b]; IF dist[next_bin] == ∞: RETURN (None, ∞, expanded_total)
    seg ← reconstruct_path(prev, current, next_bin)
    path.extend(seg[1:]); cost_total += dist[next_bin]; current ← next_bin; remaining.remove(next_bin)
  IF TSP:
    dist, prev, expanded ← dijkstra_greedy(graph, current); expanded_total += expanded
    back_seg ← reconstruct_path(prev, current, start); path.extend(back_seg[1:]); cost_total += dist[start]
  RETURN (path, cost_total, expanded_total)
END
```

3. Thuật toán

3.2 Greedy Best-First Search Algorithm

Ouput:

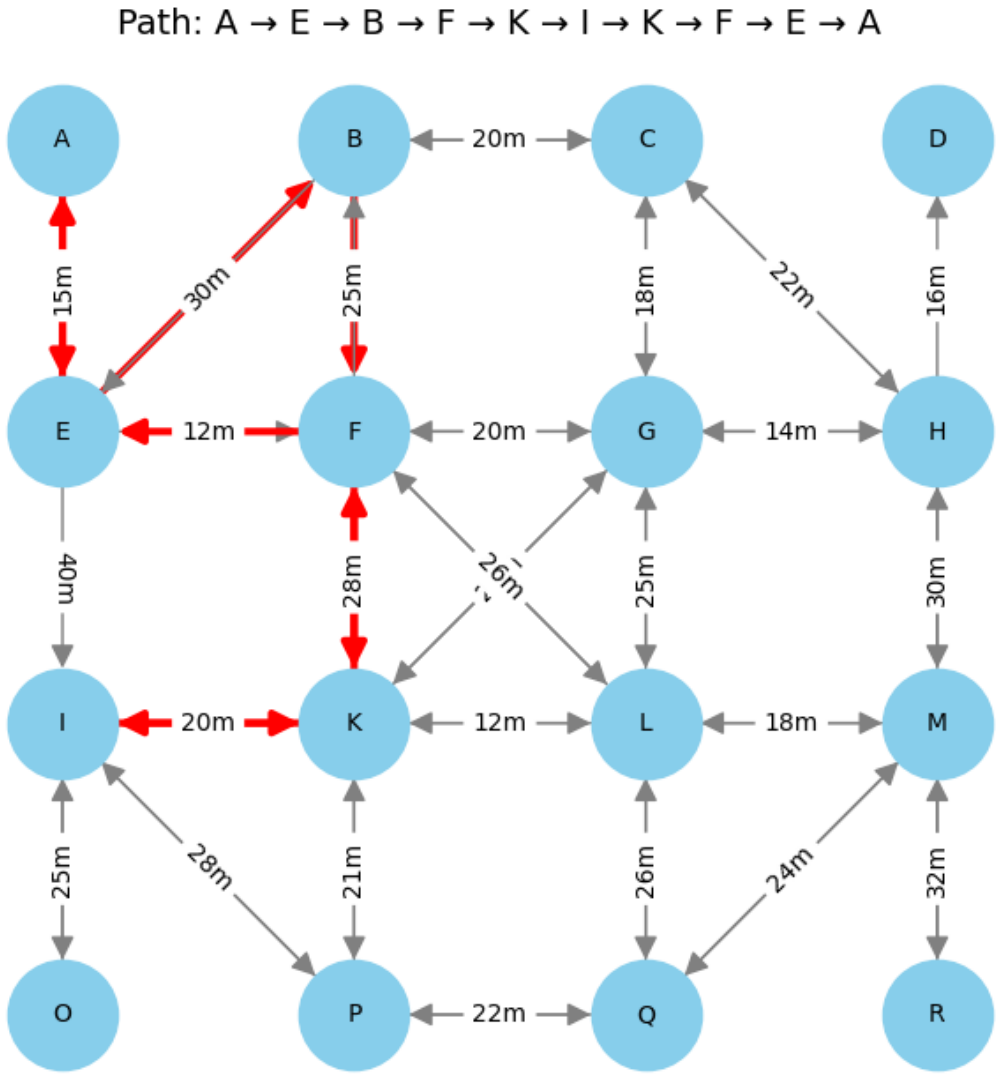
```
# ví dụ: start từ "A", thùng rác = ["B", "I", "K"]
start = "A"
bins = ["B", "I", "K"]

path, cost, expand_node = greedy_collect_bins(graph, start, bins, TSP=True)
print("Đường đi (Greedy):", path)
print("Tổng chi phí:", cost)
print("Số node mở rộng:", expand_node)

draw_graph(graph, path, size_g = 6)
```

✓ 0.9s

```
Đường đi (Greedy): ['A', 'E', 'B', 'F', 'K', 'I', 'K', 'F', 'E', 'A']
Tổng chi phí: 193
Số node mở rộng: 64
```



3. Thuật toán

3.3. A* Algorithm

Nguyên lý hoạt động	• Thuật toán sẽ tính tổng chi phí quá khứ và chi phí tương lai f(n) để đánh giá mỗi node. Thuật toán sẽ chọn nút có giá trị f(n) nhỏ nhất để mở rộng. • Công thức: f(n) = g(n) + h(n) • Hàm heuristic được sử dụng là MST + min in/out edges • Công thức: h(n) = MST(bins) + min(n -> bins) + min(bins -> start)
Đặc điểm	• Heuristic dựa trên Minimum Spanning Tree (MST) được dùng khi bài toán có nhiều mục tiêu (multi-goal path). • Cân bằng giữa tốc độ và độ chính xác vì có thể sử dụng linh hoạt các dạng heuristic • Tuy nhiên, thời gian tăng nhanh với số node lớn, và phụ thuộc vào chất lượng heuristic.
Độ phức tạp	• Thời gian: O(n! * log(n!)) (Phụ thuộc vào độ chính xác của heuristic) • Bộ nhớ: Trung bình

3. Thuật toán

3.3. A* Algorithm

Mã giả

```
FUNCTION AStarCollect(graph, start, bins, TSP=False):
  bins ← set(bins); IF TSP: bins ← bins ∪ {start}
  rem0 ← frozenset(bins - {start})
  pq ← [(h(start, rem0), 0, start, rem0, [start])]  # (f, g, curr, rem, path)
  best_g ← {}; expanded ← 0
  WHILE pq not empty:
    f, g, curr, rem, path ← heapq.pop(pq); expanded += 1
    IF rem empty:
      IF NOT TSP: RETURN (path, g, expanded)
      cost_back, back_path ← shortest_path(graph, curr, start)
      RETURN (path + back_path[1:], g + cost_back, expanded)
    IF (curr, rem) in best_g AND best_g[(curr, rem)] ≤ g: CONTINUE
    best_g[(curr, rem)] ← g
    FOR (nbr, info) IN graph[curr]:
      g2 ← g + info["distance"]
      rem2 ← frozenset(rem - {nbr})
      f2 ← g2 + h(nbr, rem2)
      heapq.push(pq, (f2, g2, nbr, rem2, path + [nbr]))
  RETURN (None, ∞, expanded)
END
```

3. Thuật toán

3.3. A* Algorithm

Ouput:

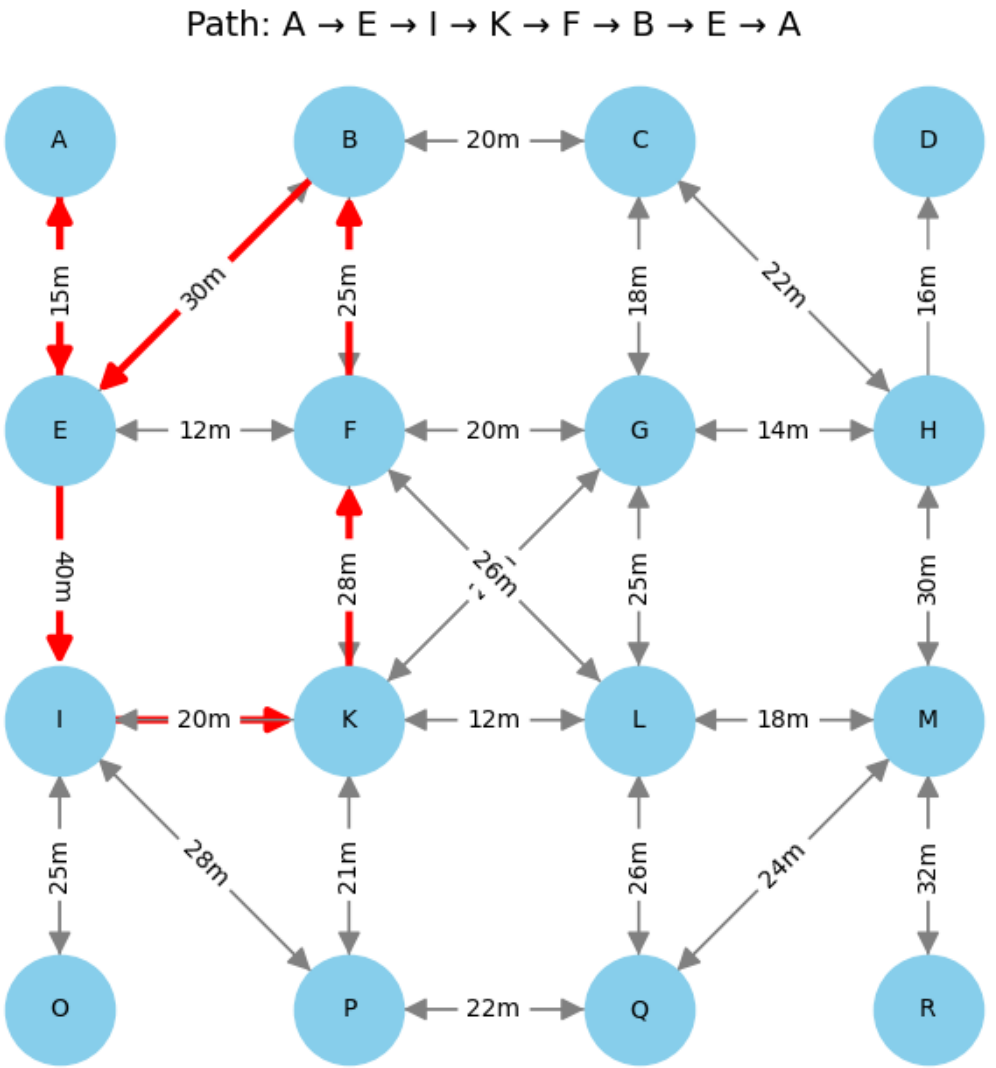
```
# ví dụ: start từ "B", thùng rác = ["B", "I", "K"]
start = "A"
bins = ["B", "I", "K"]

path, cost, expand_node = a_star_collect(graph, start, bins, TSP=True)
print("Đường đi (A*):", path)
print("Tổng chi phí:", cost)
print("Số node mở rộng:", expand_node)

draw_graph(graph, path, size_g = 6)
```

✓ 0.6s

```
Đường đi (A*): ['A', 'E', 'I', 'K', 'F', 'B', 'E', 'A']
Tổng chi phí: 173
Số node mở rộng: 11
```



4. Thử nghiệm, kết quả và đánh giá

Business AI Lab

2025-10

4. Thử nghiệm, kết quả và đánh giá

4.1. Thử nghiệm và kết quả:

Input:
Bắt đầu: “A”
Số node bins tăng dần lần lượt là 2, 4, 6, 8.
Comeback “A”: True

Output:
Cost
Running_Time (s)
Expanded_Nodes

BẢNG SO SÁNH CÁC THUẬT TOÁN

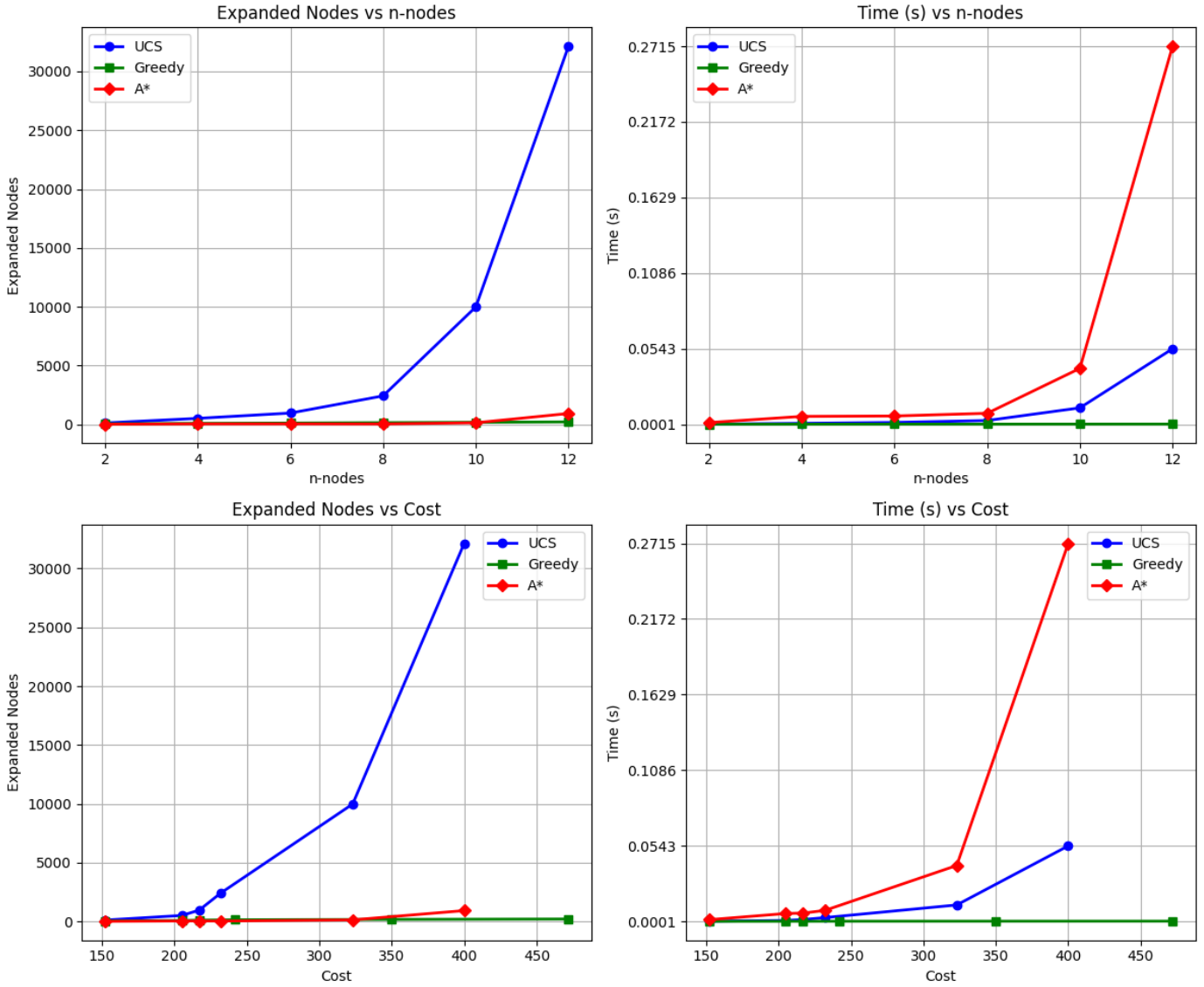
n	Algorithm	Cost	Time (s)	Expanded Nodes
2	UCS	152	0.000120	116
2	Greedy	152	0.000050	48
2	A*	152	0.001218	12
4	UCS	205	0.000679	510
4	Greedy	205	0.000081	80
4	A*	205	0.005665	41
6	UCS	217	0.001317	960
6	Greedy	217	0.000088	112
6	A*	217	0.005960	43
8	UCS	232	0.002711	2421
8	Greedy	242	0.000120	144
8	A*	232	0.007891	35
10	UCS	323	0.011796	9982
10	Greedy	350	0.000142	176
10	A*	323	0.040141	134
12	UCS	400	0.054237	32140
12	Greedy	472	0.000167	208
12	A*	400	0.271525	924

4. Thử nghiệm, kết quả và đánh giá

4.1. Thử nghiệm và kết quả:

Input:
Bắt đầu: “A”
Số node bins tăng dần lần lượt là 2, 4, 6, 8.
Comeback “A”: True

Output:
Cost
Running_Time (s)
Expanded_Nodes



4. Thử nghiệm, kết quả và đánh giá

4.2. Đánh giá:

Dựa trên thử nghiệm:

- **Greedy** cho ra kết quả nhanh nhất, mất ít tài nguyên để tính toán nhưng chi phí kém tối ưu
- **UCS** cho ra chi phí tối ưu, mất ít thời gian hơn nhưng cực kì tốn tài nguyên tính toán khi quy mô lớn
- **A*** cũng cho ra được chi phí tối ưu, mất ít tài nguyên nhưng lại rất tốn thời gian khi số nút thấp do mất thời gian cho việc tính heuristic. Nhưng với quy mô lớn, thời gian của A* sẽ được cải thiện hơn so với UCS

4. Thử nghiệm, kết quả và đánh giá

4.3. Kết luận:

- Khi bài toán có quy mô nhỏ, UCS và Greedy đều cho kết quả nhanh.
- Tuy nhiên, khi quy mô tăng, A* vượt trội hơn cả về **hiệu năng** và **chất lượng lời giải**, do biết kết hợp giữa chi phí thực tế và ước lượng heuristic.

Số node	Algorithm	Thời gian	Bộ nhớ	Tối ưu
Ít	UCS	★★	★★	★★★
	Greedy	★★★	★★★	★
	A*	★	★★★	★★★
Nhiều	UCS	★	★	★★★
	Greedy	★★★	★★★	★
	A*	★★	★★★	★★★

5. Ứng dụng thực tế

Business AI Lab

2025-10

5. Ứng dụng thực tế

5.1 Uniform Cost Search Algorithm

- Khi không gian tìm kiếm nhỏ thì thời gian xử lý cũng không quá lớn.

- Phương pháp sử dụng UCS là phù hợp nhất vì không phải tính heuristic, luôn đảm bảo đường đi tối ưu mà không tốn quá nhiều thời gian thực thi

- Ví dụ thực tế: Xe thu gom rác trong khu nhỏ; robot trong nhà máy nhỏ

5. Ứng dụng thực tế

5.2 Greedy Best-First Search Algorithm

- Khi lượng node lớn: việc tìm kiếm toàn bộ không gian trở nên rất tốn kém.
- Greedy Best-First Search là lựa chọn hợp lý hơn vì:

  + Sử dụng heuristic để ưu tiên hướng đi có vẻ “gần đích” nhất

  + Giảm mạnh số node mở rộng và rút ngắn thời gian tìm kiếm.

- Tuy nhiên, Greedy không đảm bảo tối ưu – kết quả chỉ là đường đi xấp xỉ tốt nhất trong thời gian ngắn
- Ví dụ thực tế: Ứng dụng bản đồ, drone thành phố, robot hút bụi tòa nhà

5. Ứng dụng thực tế

5.3 A* Algorithm

- Khi lượng node trung bình: Khi số node tăng lên, không gian tìm kiếm mở rộng theo cấp số nhân.

- Thuật toán A* với heuristic mạnh (MST + Dijkstra) sẽ là lựa chọn tốt nhất vì:

  + kết hợp giữa chi phí thực (g(n)) và chi phí ước lượng (h(n))

  + giảm đáng kể số node phải mở rộng, đảm bảo đường đi tối ưu và thời gian xử lý hợp lý

- Ví dụ thực tế: Giao hàng nội thành, robot vận chuyển, drone giao thuốc

6. Mở rộng

Business AI Lab

6. Mở rộng

Một vài hàm heuristic khác cho Greedy

Heuristic	Ý tưởng	Độ phức tạp	Ưu điểm	Nhược điểm	Độ chặt
Nearest Neighbor (NN)	Luôn chọn thành phố gần nhất chưa thăm	$O(n^2)$	Rất nhanh, dễ cài đặt	Dễ mắc local optima, không quay lại tối ưu	Yếu
Cheapest Insertion	Bắt đầu từ 2 node, chèn node vào vị trí có chi phí tăng nhỏ nhất	$O(n^3)$	Tour tốt hơn NN	Tốn thời gian hơn	Khá
Nearest Insertion	Chọn node gần tour nhất rồi chèn vào vị trí tối ưu	$O(n^3)$	Cân bằng giữa NN & Insertion	Chậm hơn NN	Trung bình khá
Savings (Clarke–Wright)	Ghép cặp điểm dựa trên mức tiết kiệm khi đi chung tuyến	$O(n^2 \log n)$	Hiệu quả với cấu trúc cụm	Cần điểm xuất phát rõ ràng	Khá
Cheapest Link (Edge-based Greedy)	Chọn cạnh rẻ nhất chưa tạo chu trình con	$O(n^2 \log n)$	Tour khá tốt, logic đơn giản	Kiểm tra chu trình con phức tạp	Khá
Randomized Greedy (GRASP)	Thêm ngẫu nhiên vào top-k lựa chọn tốt nhất, rồi local search	tùy ($\approx m \times N$ N)	Giảm local optima, cải thiện kết quả	Cần lặp nhiều, tuning tham số	Tốt nếu lặp đủ

6. Mở rộng

Một vài hàm heuristic khác cho A*

Heuristic	Ý tưởng	Độ phức tạp	Ưu điểm	Nhược điểm	Đánh giá độ chặt
MST (Minimum Spanning Tree)	Ước lượng chi phí nhỏ nhất để nối các node chưa thăm bằng cây khung nhỏ nhất	$O(n^2 \log n)$	Dễ cài, admissible, chạy nhanh	Bound khá lỏng, chưa tính chi phí ra/vào node hiện tại	Trung bình
MST + min in/out edges	MST + chi phí rời khỏi node hiện tại và quay về điểm xuất phát	$O(n^2 \log n)$	Bound chặt hơn MST, vẫn nhanh	Cần tính thêm 2 cạnh min	Khá
1-Tree (Held–Karp bound)	Tạo MST trên n−1 node + 2 cạnh nhỏ nhất nối node còn lại	$O(n^2 \log n)$	Bound rất chặt, hiệu quả trong thực tế	Tính toán phức tạp hơn	Rất khá
Minimum Matching (Christofides bound)	Ghép cặp các node bậc lẻ trong MST để tạo tour gần tối ưu	$O(n^3)$	Bound tốt, logic đẹp	Nặng với n lớn, phức tạp cài đặt	Rất khá
Assignment (Hungarian Lower Bound)	Giải bài toán gán chi phí tối thiểu giữa các node chưa thăm	$O(n^3)$	Bound chặt nhất, gần với nghiệm tối ưu	Tính toán nặng, khó tích hợp online	Rất tốt
Convex Hull + MST inside	Chu vi bao ngoài + cây khung nhỏ nhất cho điểm bên trong	$O(n \log n)$	Dễ hiểu, phù hợp dữ liệu tọa độ	Bound không thật chặt với phân bố ngẫu nhiên	Trung bình
Simple min distance sum	Cộng các khoảng cách nhỏ nhất từ node hiện tại đến node chưa thăm	$O(n^2)$	Rất nhanh, dễ tính	Bound yếu, không phản ánh cấu trúc tour	Yếu

7. Thảo luận

Business AI Lab

Thank you!

You're now ready to explore the exciting world of AI!

ENGLISH

ENGLISH

ENGLISH

Optimal garbage collection problem

Team 2

Nguyễn Ngân An - 11247253

Phạm Hữu Gia Ân - 11247254

Nguyễn Trọng Đại - 11247268

Lê Bá Phong - 11247339

Instructor: Dr. Nguyen Trong Nghia

Business AI Lab

1. Problems in practice

Business AI Lab

2025-10

1. Problems in practice

1.1. Garbage collection problem

a. Problem description

Every day, garbage trucks need to pass dozens to hundreds of trash cans scattered throughout the neighborhood and return to the station via the shortest route.

Find the optimal path to help:

+ Fuel economy
+ Reduce operating costs
+ Increase collection efficiency, …

b. Problem modeling

- Modeling in problem form:
  + Travelling Salesman Problem – TSP
  + Hamiltonian Path Problem – HPP

1. Problems in practice

1.2. Other real-life situations

Shipper delivers goods

Tourists go sightseeing

Transport robots

2. Formula Problem

Business AI Lab

2. Formula Problem

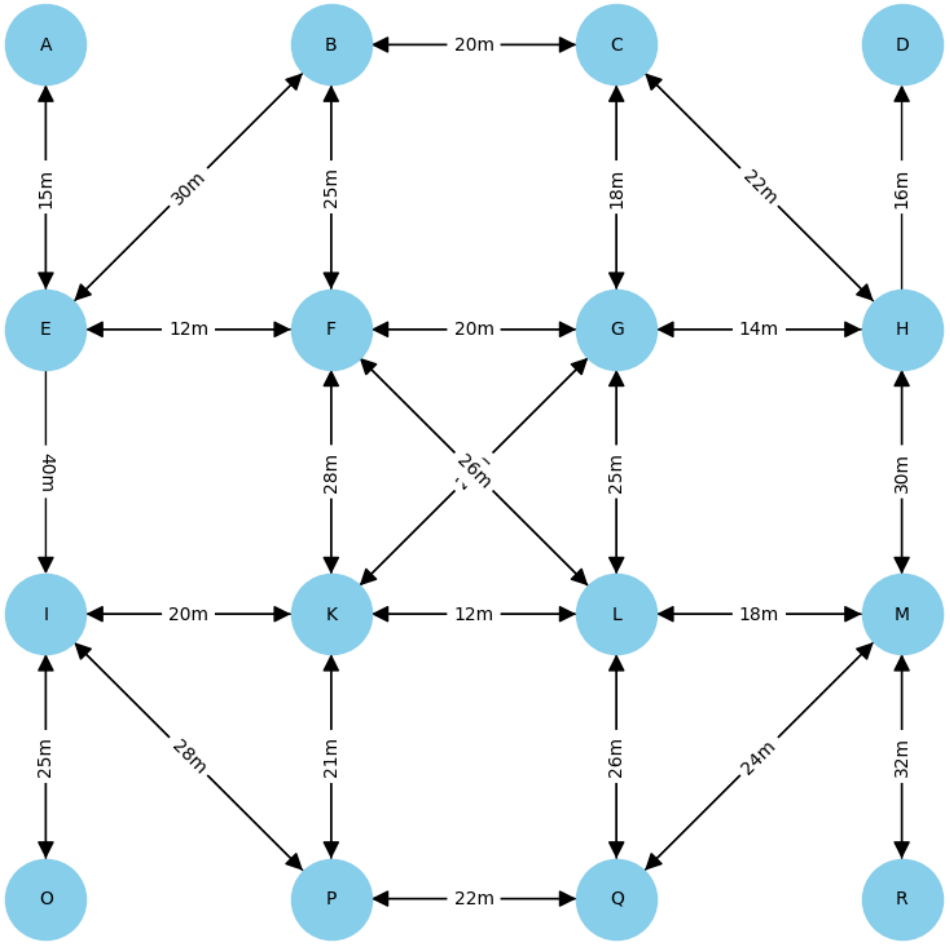
2.1. Map Simulation

Description:

- Each node is an area to traverse
- The connecting edge between the 2 nodes is the garbage collection road

A few other factors:

- One-way road: E -> I
- Dead end (returnable): A, O, R
- Dead end (irreversible): D



2. Problem formula

2.2. Problem model

Ingredient	Description
Status	(Current city, set of cities passed through)
Start Status	(A, {A})
Goal	(A, {B, C, D, E, ...}) – visited all and returned to A
Operator	Moving to an unvisited city
Cost of the road	Total distance traveled
Heuristic	Estimated cost to unvisited nodes

3. Algorithm

1. Uniform Cost Search Algorithm
2. Greedy Best-First Sreach Algorithm
3. A* Algorithm

Business AI Lab

2025-10

3. Algorithm

3.1. Uniform Cost Search Algorithm - UCS

Principle of operation	• Start at the root node and continue by browsing the next nodes with the lowest weighting or cost from the root node. • The algorithm chooses to expand the node with the smallest total cost from the root to that node (g(n))
Character	• UCS does not use heuristics. • Make sure to find the path with the least cost if all costs are close to ≥ 0. • Equivalent to Dijkstra's Algorithm when applied to weighted graphs
Complexity	• Time: O(b^(1+C*/ε)) (where b is the branching factor, C* is the optimal cost, ε is the smallest edge cost). • Memory: Proportional to the number of expanded nodes. (quite high when the number of nodes is large)

3. Algorithm

3.1. Uniform Cost Search Algorithm - UCS

Pseudocode

```
FUNCTION uniform_cost_search(graph, start, bins, TSP=False)
  bins ← set(bins); IF TSP: bins ← bins ∪ {start}
  pq ← min-heap [(0, start, frozenset(bins - {start}), [start])]
  visited ← {} ; expanded ← 0
  WHILE pq not empty:
    g, curr, remaining, path ← heapq.pop(pq); expanded += 1
    IF remaining empty:
      IF NOT TSP: RETURN (path, g, expanded)
      back_cost, back_path ← shortest_path(graph, curr, start)
      IF back_path: RETURN (path + back_path[1:], g + back_cost, expanded)
    IF (curr, remaining) in visited AND visited[(curr,remaining)] ≤ g: CONTINUE
    visited[(curr,remaining)] ← g
    FOR neighbor, info IN graph[curr].items():
      new_g ← g + info["distance"]
      new_rem ← frozenset(set(remaining) - {neighbor})
      heapq.push(pq, (new_g, neighbor, new_rem, path + [neighbor]))
  RETURN (None, ∞, expanded)
END
```

3. Algorithm

3.2. Greedy Best-First Search Algorithm

Principle of operation	• The algorithm will use 1 heuristic evaluation function h(n) to go from the current node to the nearest node and repeat until it reaches the last node. • The heuristic function used is Dijkstra: Calculate the estimated cost from the current node to the bins and go to the nearest bins, repeating until it goes all the way back.
Character	• Search is faster than UCS because it doesn't care about the actual cost (g(n)). • Optimum is not guaranteed. • Suitable when fast results are needed or in large search spaces.
Complexity	• Time: Strongly depends on the quality of the heuristic. • Memory: Low

3. Algorithm

3.2 Greedy Best-First Search Algorithm

Pseudocode

```
FUNCTION greedy_collect_bins(graph, start, bins, TSP=False)
  current ← start; path ← [start]; cost_total ← 0; remaining ← set(bins); expanded_total ← 0
  WHILE remaining:
    dist, prev, expanded ← dijkstra_greedy(graph, current); expanded_total += expanded
    next_bin ← argmin_{b∈remaining} dist[b]; IF dist[next_bin] == ∞: RETURN (None, ∞, expanded_total)
    seg ← reconstruct_path(prev, current, next_bin)
    path.extend(seg[1:]); cost_total += dist[next_bin]; current ← next_bin; remaining.remove(next_bin)
  IF TSP:
    dist, prev, expanded ← dijkstra_greedy(graph, current); expanded_total += expanded
    back_seg ← reconstruct_path(prev, current, start); path.extend(back_seg[1:]); cost_total += dist[start]
  RETURN (path, cost_total, expanded_total)
END
```

3. Algorithm

3.3. A* Algorithm

Principle of operation	• The algorithm calculates the sum of past costs and future costs f(n) to evaluate each node. The algorithm will select the node with the smallest f(n) value to expand. • Formula: f(n) = g(n) + h(n) • The heuristic function used is MST + min in/out edges • Formula: h(n) = MST(bins) + min(n -> bins) + min(bins -> start)
Character	• Heuristics based on the Minimum Spanning Tree (MST) are used when the problem has a multi-goal path. • Balance between speed and accuracy as heuristic forms can be used flexibly • However, the time increases rapidly with the large number of nodes, and depends on the quality of the heuristic.
Complexity	• Time: depends on the accuracy of the heuristic • Memory: Medium

3. Thuật toán

3.3. A* Algorithm

Pseudocode

```
FUNCTION AStarCollect(graph, start, bins, TSP=False):
  bins ← set(bins); IF TSP: bins ← bins ∪ {start}
  rem0 ← frozenset(bins - {start})
  pq ← [(h(start, rem0), 0, start, rem0, [start])]  # (f, g, curr, rem, path)
  best_g ← {}; expanded ← 0
  WHILE pq not empty:
    f, g, curr, rem, path ← heapq.pop(pq); expanded += 1
    IF rem empty:
      IF NOT TSP: RETURN (path, g, expanded)
      cost_back, back_path ← shortest_path(graph, curr, start)
      RETURN (path + back_path[1:], g + cost_back, expanded)
    IF (curr, rem) in best_g AND best_g[(curr, rem)] ≤ g: CONTINUE
    best_g[(curr, rem)] ← g
    FOR (nbr, info) IN graph[curr]:
      g2 ← g + info["distance"]
      rem2 ← frozenset(rem - {nbr})
      f2 ← g2 + h(nbr, rem2)
      heapq.push(pq, (f2, g2, nbr, rem2, path + [nbr]))
  RETURN (None, ∞, expanded)
END
```

4. Testing, results, and evaluation

Business AI Lab

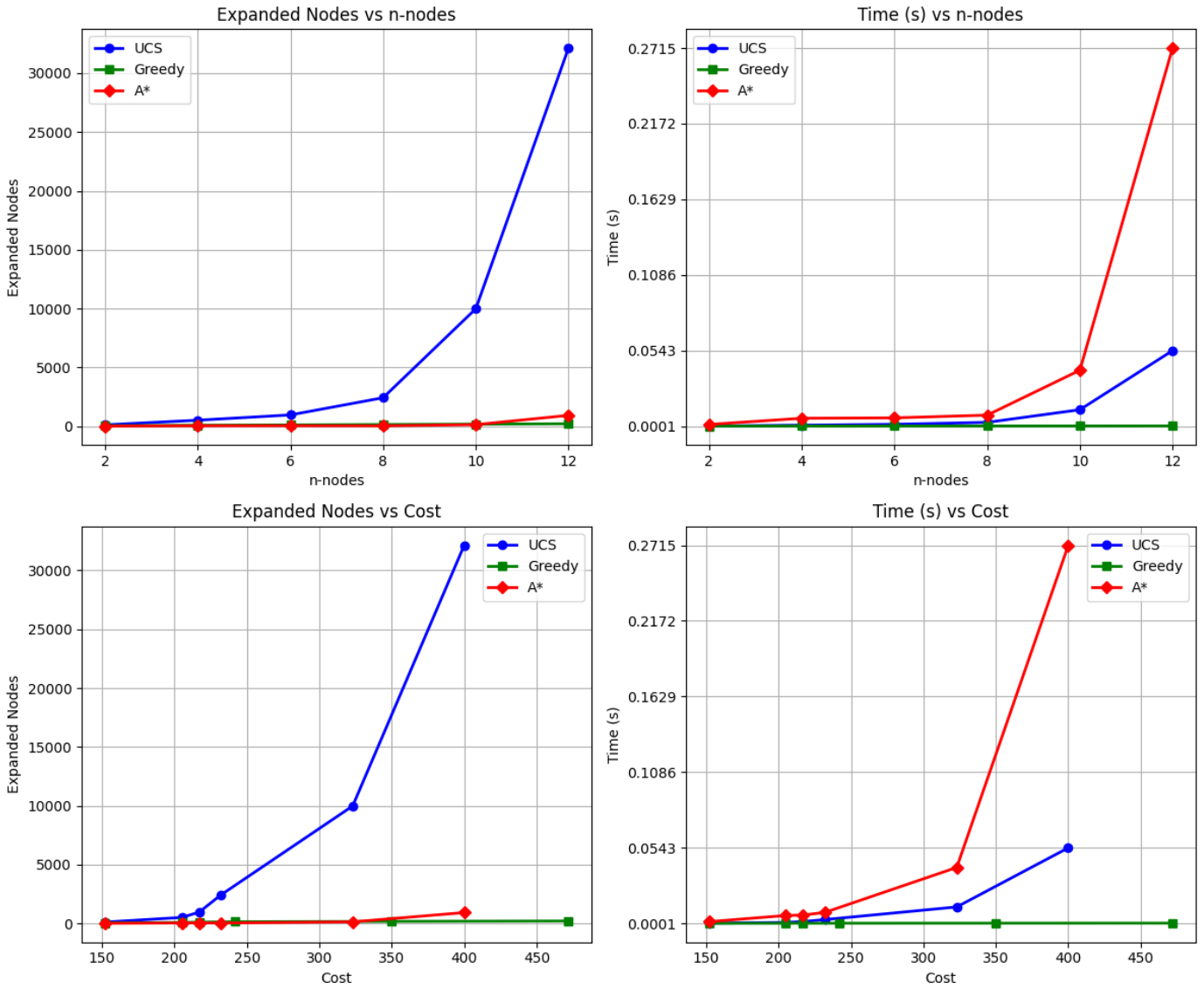
2025-10

4. Testing, results, and evaluation

4.1. Tests and results:

Input:
Start: "A"
The number of node bins gradually increases to 2, 4, 6, 8, respectively.
Comeback "A": True

Output:
Cost_of_road
Running_Time(s)
Expanded_Nodes



4. Testing, results, and evaluation

4.2. Evaluation:

Test-based:

- **Greedy** produces the fastest results, takes few resources to calculate, but is less cost-optimal.
- **UCS** provides optimal cost, takes less time but is extremely computationally resource-intensive when the number of nodes is large
- **A*** also provides optimal costs, consumes few resources, but is very time-consuming when the number of nodes is low due to the time spent on heuristic calculations. But when the number of nodes is large, the time of A* will be improved compared to UCS

4. Testing, results, and evaluation

4.3. Conclusion:

- When the problem is small, UCS and Greedy both give fast results.
- However, as the scale increases, A* excels in both performance and solution quality, due to the combination of actual cost and heuristic estimation.

Number of nodes	Algorithm	Time	Memory	Optimization
Few	UCS	★★	★★	★★★
	Greedy	★★★	★★★	★
	A*	★	★★★	★★★
Many	UCS	★	★	★★★
	Greedy	★★★	★★★	★
	A*	★★	★★★	★★★

5. Practical application

Business AI Lab

2025-10

5. Practical application

5.1 Uniform Cost Search Algorithm

- Khi không gian tìm kiếm nhỏ thì thời gian xử lý cũng không quá lớn.

- Phương pháp sử dụng UCS là phù hợp nhất vì không phải tính heuristic, luôn đảm bảo đường đi tối ưu mà không tốn quá nhiều thời gian thực thi

- Ví dụ thực tế: Xe thu gom rác trong khu nhỏ; robot trong nhà máy nhỏ

5. Practical application

5.2 Greedy Best-First Search Algorithm

- Khi lượng node lớn: việc tìm kiếm toàn bộ không gian trở nên rất tốn kém.
- Greedy Best-First Search là lựa chọn hợp lý hơn vì:

  + Sử dụng heuristic để ưu tiên hướng đi có vẻ “gần đích” nhất

  + Giảm mạnh số node mở rộng và rút ngắn thời gian tìm kiếm.
- Tuy nhiên, Greedy không đảm bảo tối ưu – kết quả chỉ là đường đi xấp xỉ tốt nhất trong thời gian ngắn
- Ví dụ thực tế: Ứng dụng bản đồ, drone thành phố, robot hút bụi tòa nhà

5. Practical application

5.3 A* Algorithm

- Khi lượng node trung bình: Khi số node tăng lên, không gian tìm kiếm mở rộng theo cấp số nhân.

- Thuật toán A* với heuristic mạnh (MST + Dijkstra) sẽ là lựa chọn tốt nhất vì:

  + kết hợp giữa chi phí thực (g(n)) và chi phí ước lượng (h(n))

  + giảm đáng kể số node phải mở rộng, đảm bảo đường đi tối ưu và thời gian xử lý hợp lý

- Ví dụ thực tế: Giao hàng nội thành, robot vận chuyển, drone giao thuốc

6. Expansion

Business AI Lab

6. Expansion

A few other heuristic functions for Greedy

Heuristic	Idea	Complexity	Advantage	Disadvantage	Tightness
Nearest Neighbor (NN)	Always choose the nearest city that hasn't been visited yet	$O(n^2)$	Very fast, easy to install	Easy to suffer from local optima, not returning to optimal	Weak
Cheapest Insertion	Starting with 2 nodes, insert the node into the location with the smallest increase in cost	$O(n^3)$	Better Tour NN	More time-consuming	Fairly
Nearest Insertion	Select the node closest to the tour and insert it in the optimal location	$O(n^3)$	Balancing NN & Insertion	Slower than NN	Fair medium
Savings (Clarke–Wright)	Pair points based on the savings of sharing a route	$O(n^2 \log n)$	Efficient with cluster structure	Need a clear starting point	Fairly
Cheapest Link (Edge-based Greedy)	Choose the cheapest side that hasn't created a subcycle	$O(n^2 \log n)$	The tour is quite good, the logic is simple	Complex Subcycle Testing	Fairly
Randomized Greedy (GRASP)	Randomly add to the top k of the best choices, then do a local search	depending ($\approx m \times NN$)	Reduce local optima, improve outcomes	Need multiple repeats, tuning parameters	Good if repeated enough

6. Expansion

A few other heuristic functions for A*

Heuristic	Idea	Complexity	Advantage	Disadvantage	Tightness Rating
MST (Minimum Spanning Tree)	Estimate the smallest cost to join unvisited nodes with the smallest frame tree	$O(n^2 \log n)$	Easy to install, admissible, fast running	The bound is quite fluid, not counting the cost of entering/exiting the current node	Medium
MST + min in/out edges	MST + the cost of leaving the current node and returning to the starting point	$O(n^2 \log n)$	Bound tighter than MST, still fast	Need to calculate 2 more min edges	Fairly
1-Tree (Held–Karp bound)	Create MST on n−1 node + 2 smallest edges connecting the other node	$O(n^2 \log n)$	Very tight bound, effective in practice	More complex calculations	Very Pretty
Minimum Matching (Christofides bound)	Pair odd-tier nodes in MST to create near-optimal tours	$O(n^3)$	Good bound, beautiful logic	Heavy duty with n large, complicated installation	Very Pretty
Assignment (Hungarian Lower Bound)	Solving the problem of assigning minimum costs between unvisited nodes	$O(n^3)$	Tightest bound, closest to optimal experience	Heavy calculations, difficult to integrate online	Good
Convex Hull + MST inside	Outer envelope circumference + smallest frame tree for inner point	$O(n \log n)$	Easy to understand, suitable for coordinate data	Bound is not very tight with random distribution	Medium
Simple min distance sum	Add the smallest distances from the current node to the unvisited node	$O(n^2)$	Very fast, easy to calculate	Weak bound, does not reflect the tour structure	Weak

7. Discussion

Business AI Lab

Thank you!

You're now ready to explore the exciting world of AI!
Next Lecture: Search and Problem Solving

5. Ứng dụng

5.3 A* Algorithm