Trường Đại học Khoa học Tự nhiên
Khoa Công nghệ Thông tin

TÀI LIỆU LÝ THUYẾT TRÍ TUỆ NHÂN TẠO

Chủ đề 2

TÌM KIẾM HEURISTIC

Giảng viên: Ths. Vũ Thanh Hưng
Email: vthung@fit.hcmus.edu.vn

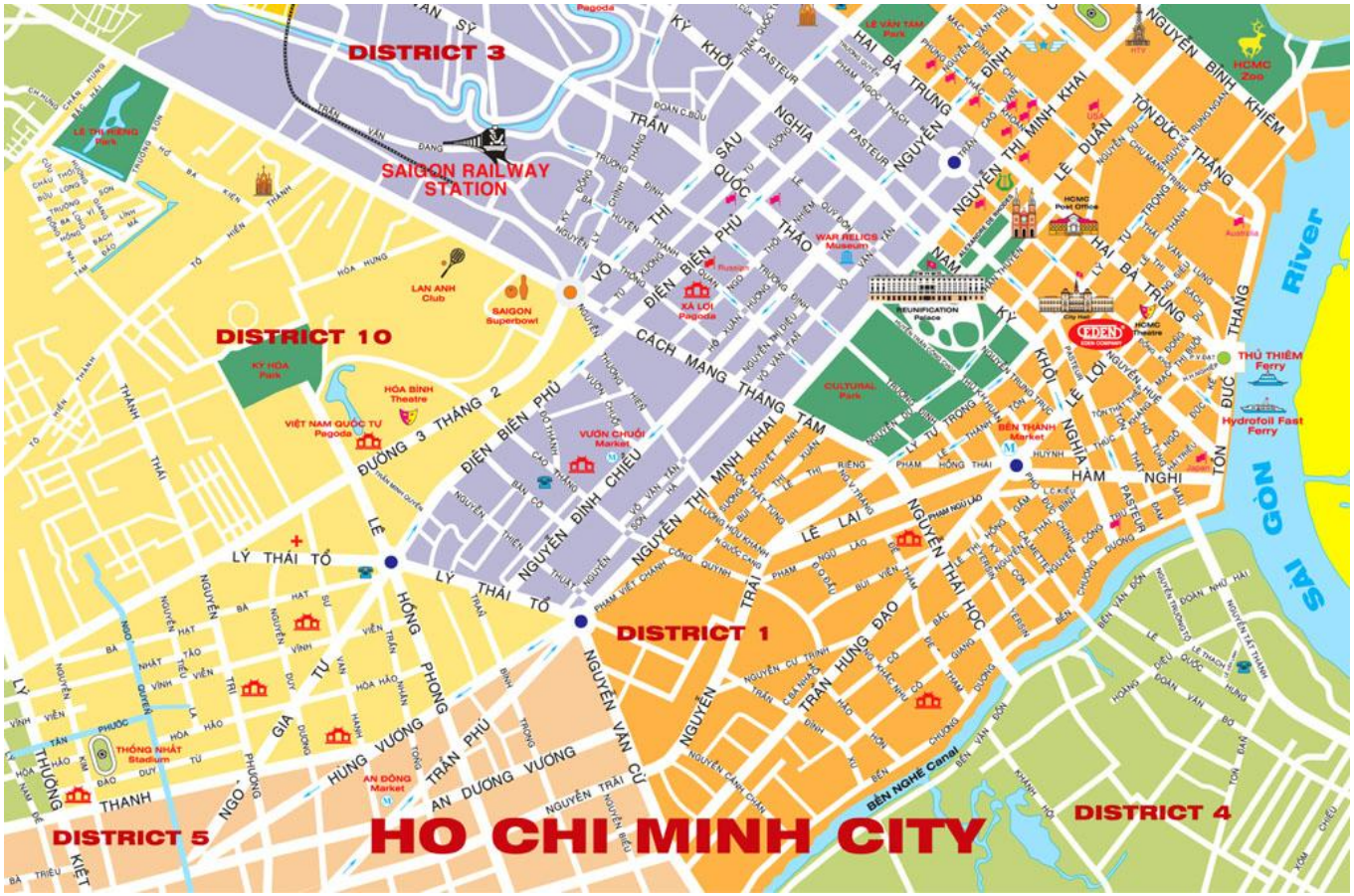
HKii/2011-2012

NỘI DUNG

- Định nghĩa heuristic
- Nhóm thuật toán tìm kiếm có heuristic
  - Tìm kiếm tốt nhất đầu tiên
  - Tìm kiếm A*
- Thuật giải leo đồi
  - Ý tưởng thuật giải leo đồi và những cải tiến
  - Ứng dụng cho bài toán tối ưu hóa
- Thuật giải di truyền
  - Sơ đồ thuật giải di truyền và các toán tử cơ bản

Người soạn: ThS. Nguyễn Ngọc Thảo

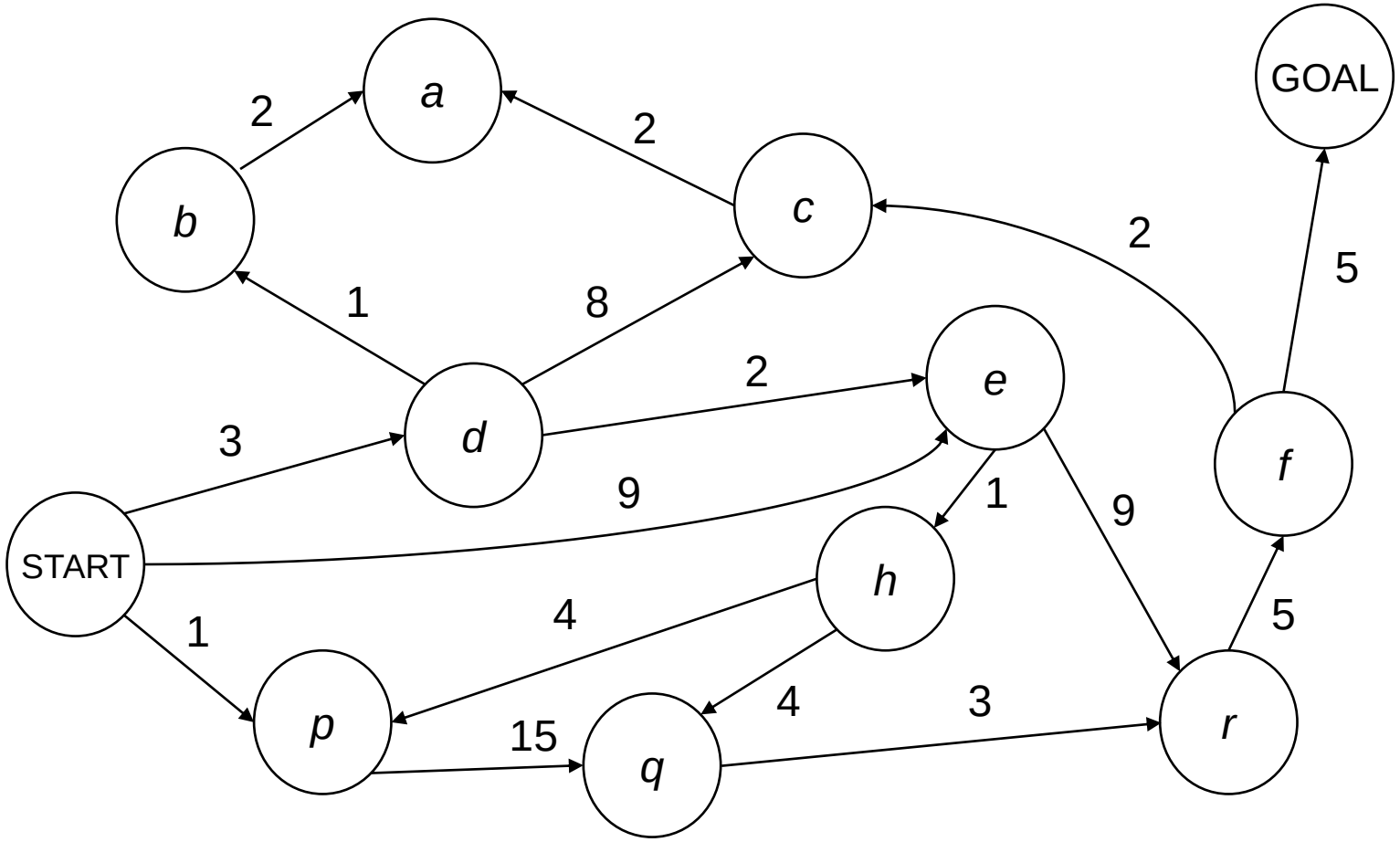
Tìm đường tp HCM



Tìm đường ngắn nhất từ trường sư phạm tới ga Sài Gòn

Người soạn: ThS. Nguyễn Ngọc Thảo

TÌM ĐƯỜNG TRÊN BẢN ĐỒ



ĐỊNH NGHĨA HEURISTIC

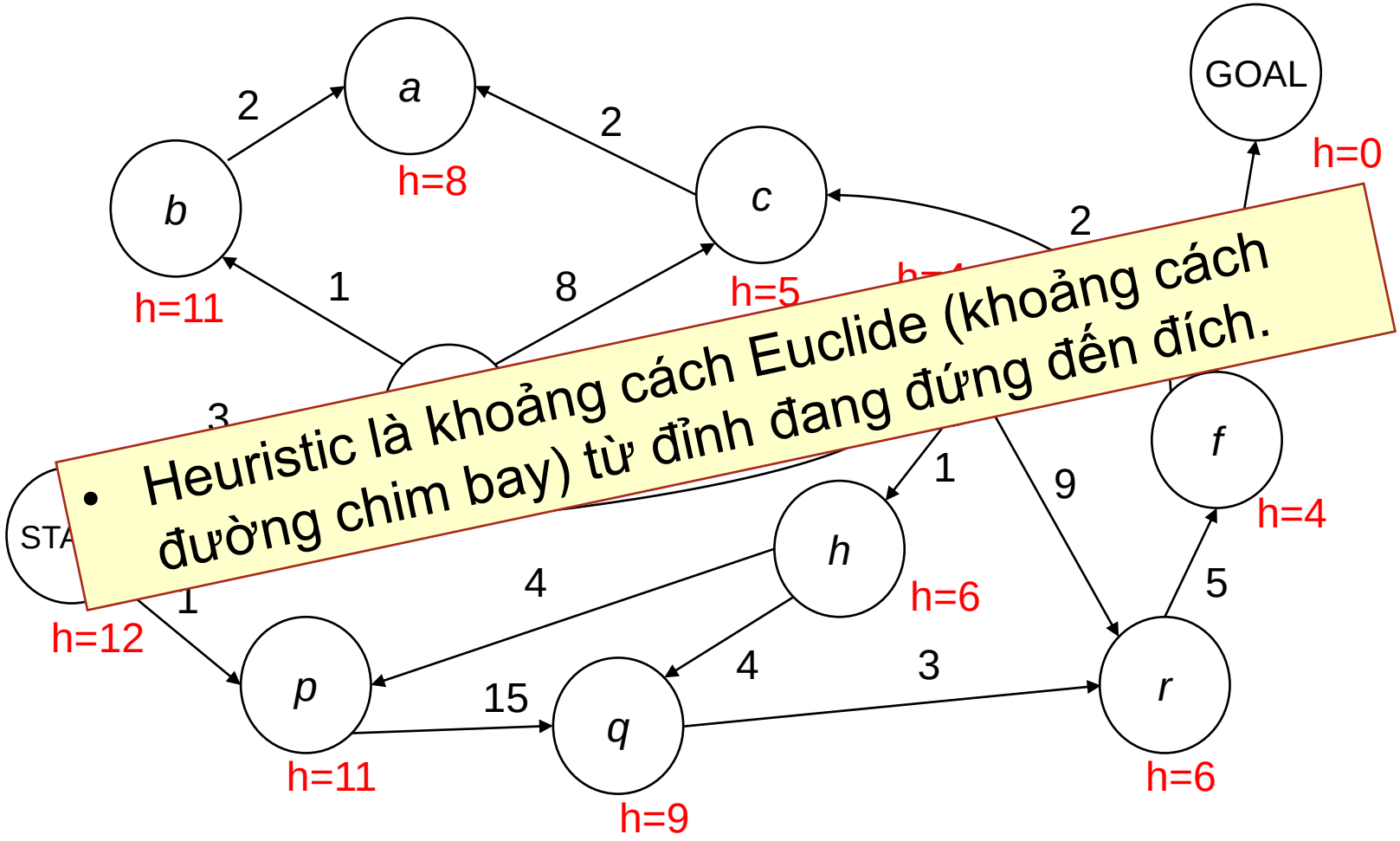
ĐỊNH NGHĨA HEURISTIC

- Giả sử bên cạnh các đặc tả chuẩn bài toán tìm kiếm, ta còn cần có một heuristic.

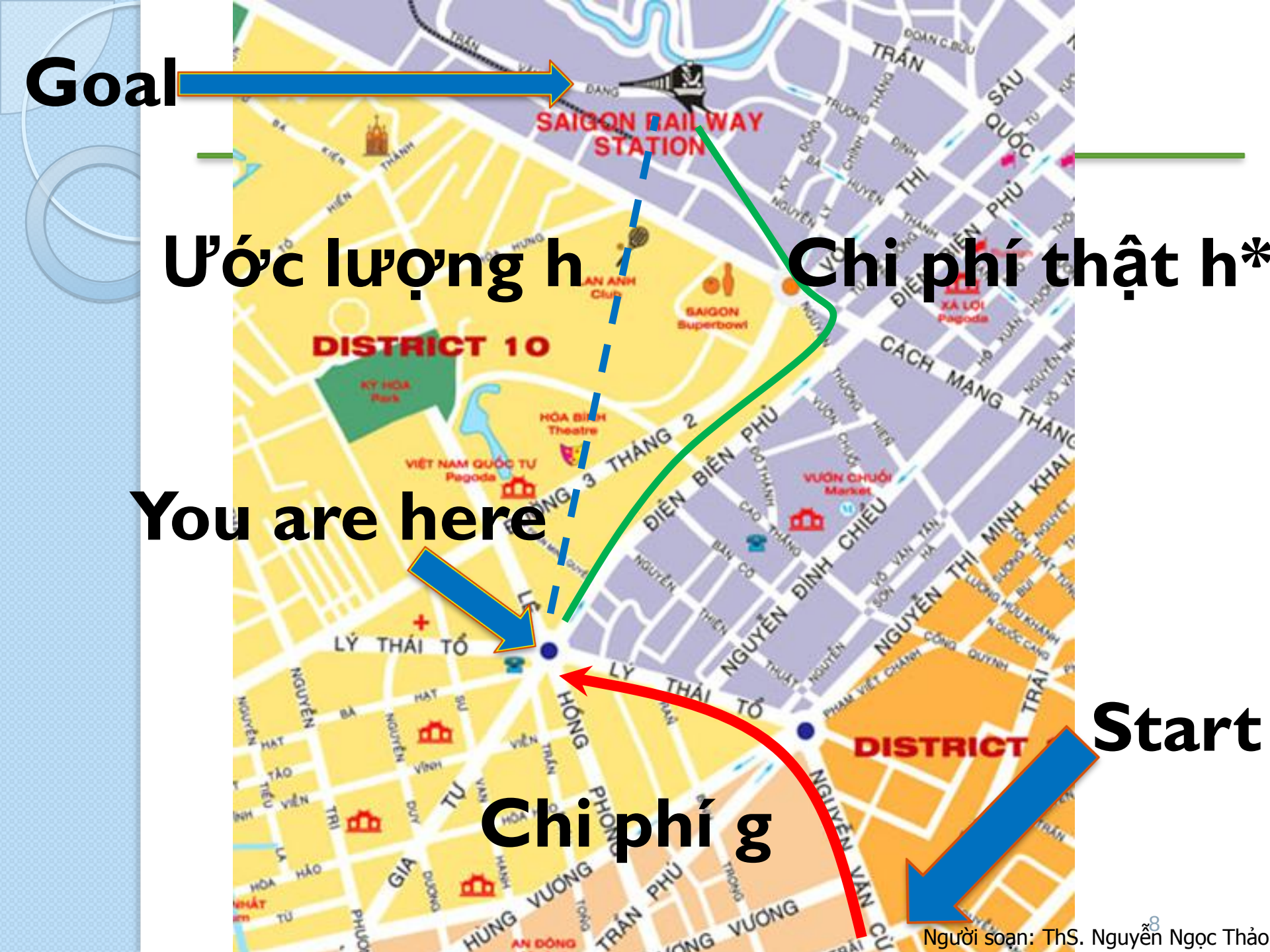
  Hàm heuristic ánh xạ một trạng thái thành một ước lượng chi phí từ đích đến trạng thái đó.

- Kí hiệu heuristic bằng hàm *h*(*s*) từ trạng thái thành giá trị chi phí.

EUCLIDEAN HEURISTIC

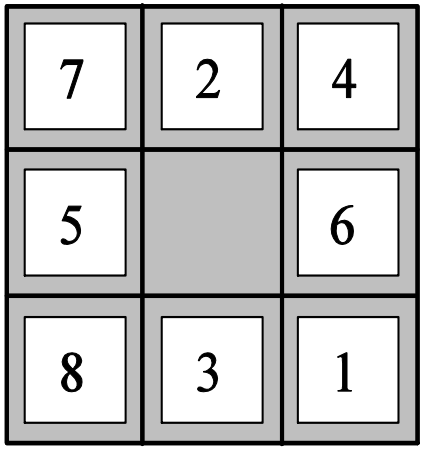


- Heuristic là khoảng cách Euclide (khoảng cách đường chim bay) từ đỉnh đang đứng đến đích.

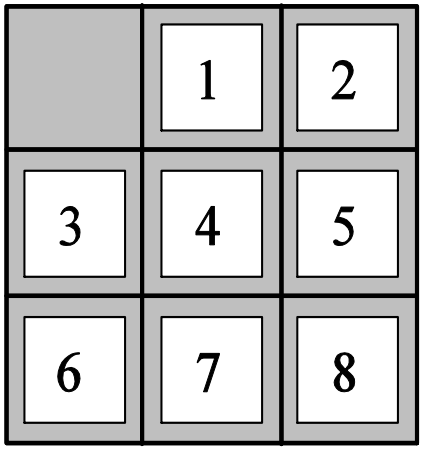

Goal
Ước lượng h
Chi phí thật h*
You are here
Start
Chi phí g
SAIGON RAILWAY STATION
DISTRICT 10
DISTRICT
LÝ THÁI TỔ
ĐIỆN BIÊN PHỦ
CÁCH MẠNG THÁNG 8
NGUYỄN ĐÌNH CHIỂU
NGUYỄN THỊ MINH KHAI
HÙNG VƯƠNG
TRẦN PHÚ
3 THÁNG 2
SAIGON Superbowl
VIỆT NAM QUỐC TỰ Pagoda
XÁ LỢI Pagoda
VƯỜN CHUỐI Market
KỲ HÒA Park
HÒA BÌNH Theatre
AN ĐÔNG

BÀI TOÁN N-PUZZLE

- N-puzzle là một bảng gỗ kích thước $N \times N$ với N-1 ô số và một ô trống. Ô số nằm cạnh ô trống có thể trượt đến ô trống.
- Mục tiêu là đẩy các ô số sao cho đạt được trạng thái đích mong muốn.



Start State



Goal State

BÀI TOÁN N-PUZZLE

- 8-puzzle có 9!/2 = 181,440 trạng thái – dễ dàng tìm được lời giải.
- 15-puzzle có khoảng 2.09×10^{13} trạng thái – tìm được lời giải tối ưu trong vài mili giây với những thuật toán tìm kiếm tốt nhất.
- 24-puzzle có khoảng 10^{25} trạng thái – chưa thể tìm được lời giải tối ưu với điều kiện máy tính và thuật toán hiện nay.

HEURISTIC TRONG 8-PUZZLE

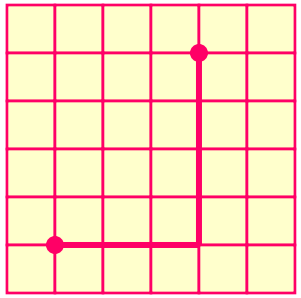
- Theo số ô nằm sai vị trí:

1		5
2	6	3
7	4	8

h = 6

1	2	3
4	5	6
7	8	

- Theo khoảng cách Manhattan:



d = dx + dy

1		5
2	6	3
7	4	8

h = 9

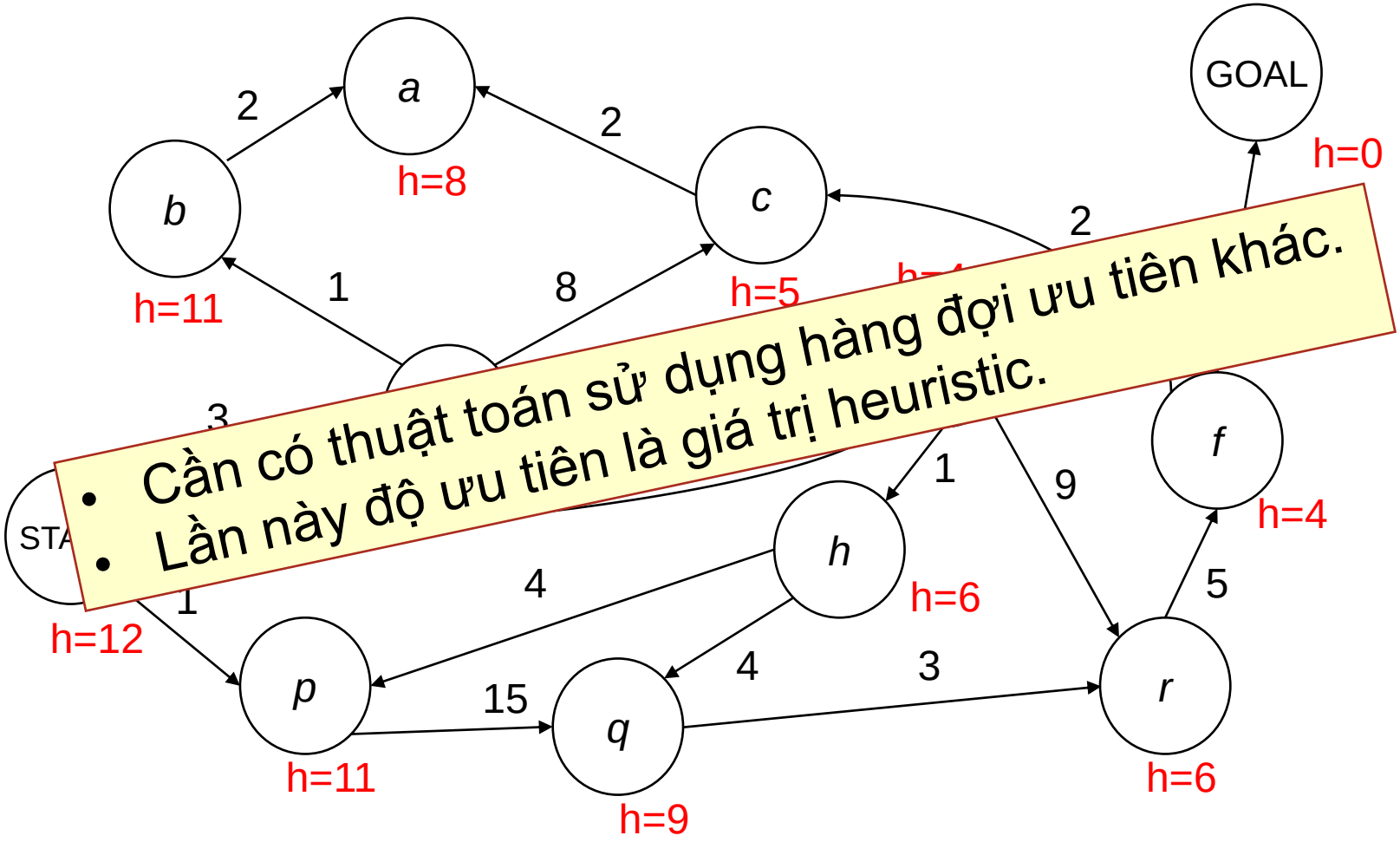
1	2	3
4	5	6
7	8	

h = 0 + 2 + 1 + 2 + 2 + 1 +0 + 1 = 9

TÌM KIẾM TỐT NHẤT ĐẦU TIÊN

(Greedy Best First Search)

EUCLIDEAN HEURISTIC



GREEDY BEST FIRST SEARCH

```
Init-PriQueue(PQ)
Insert-PriQueue(PQ,START,h(START))
while (PQ không rỗng và PQ không chứa trạng thái đích)
        (s, h) := Pop-least(PQ)
        foreach s' in succs(s)
            if s' chưa bao giờ được duyệt
                if s' chưa có trong PQ
                    Insert-PriQueue(PQ,s',h(s'))
```

- ***Thuật toán này có tìm được đường đi ngắn nhất ?***

GREEDY BEST FIRST SEARCH

Init-PriQueue(PQ)

Insert-PriQueue(PQ,START,h(START))

while (PQ không rỗng và PQ không chứa trạng thái đích)

(*s*, *h*) := Pop-least(PQ)

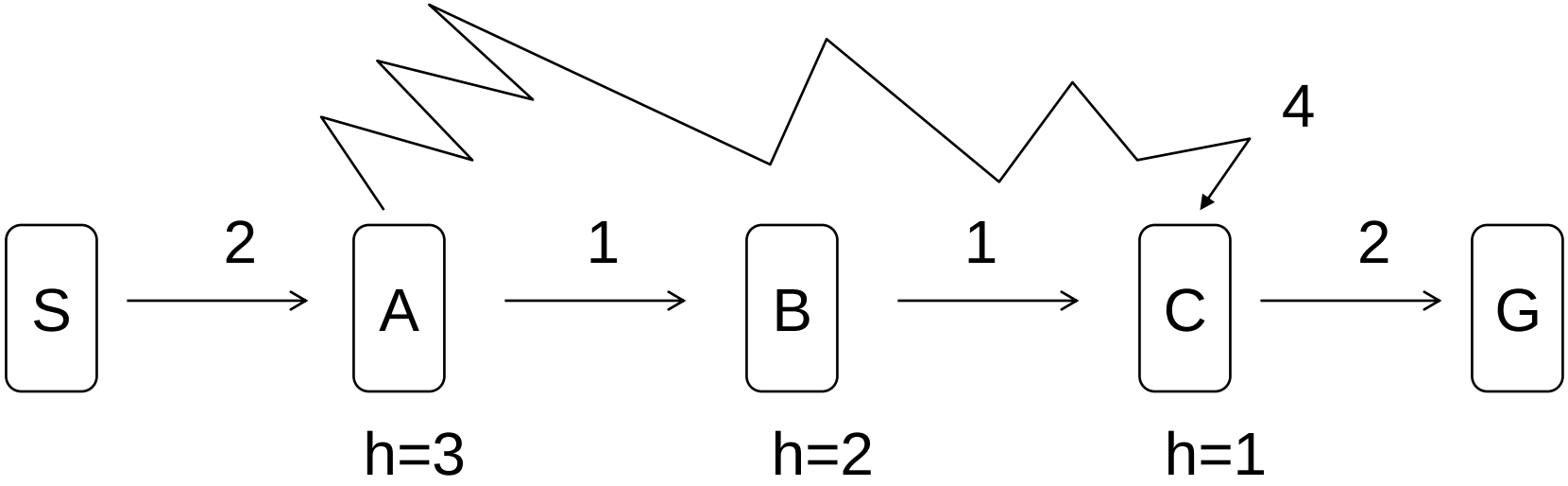
foreach *s*' in succs(*s*)

if *s*' chưa có trong PQ and *s*' chưa bao giờ được duyệt

Insert-PriQueue(PQ,*s*',h(*s*'))

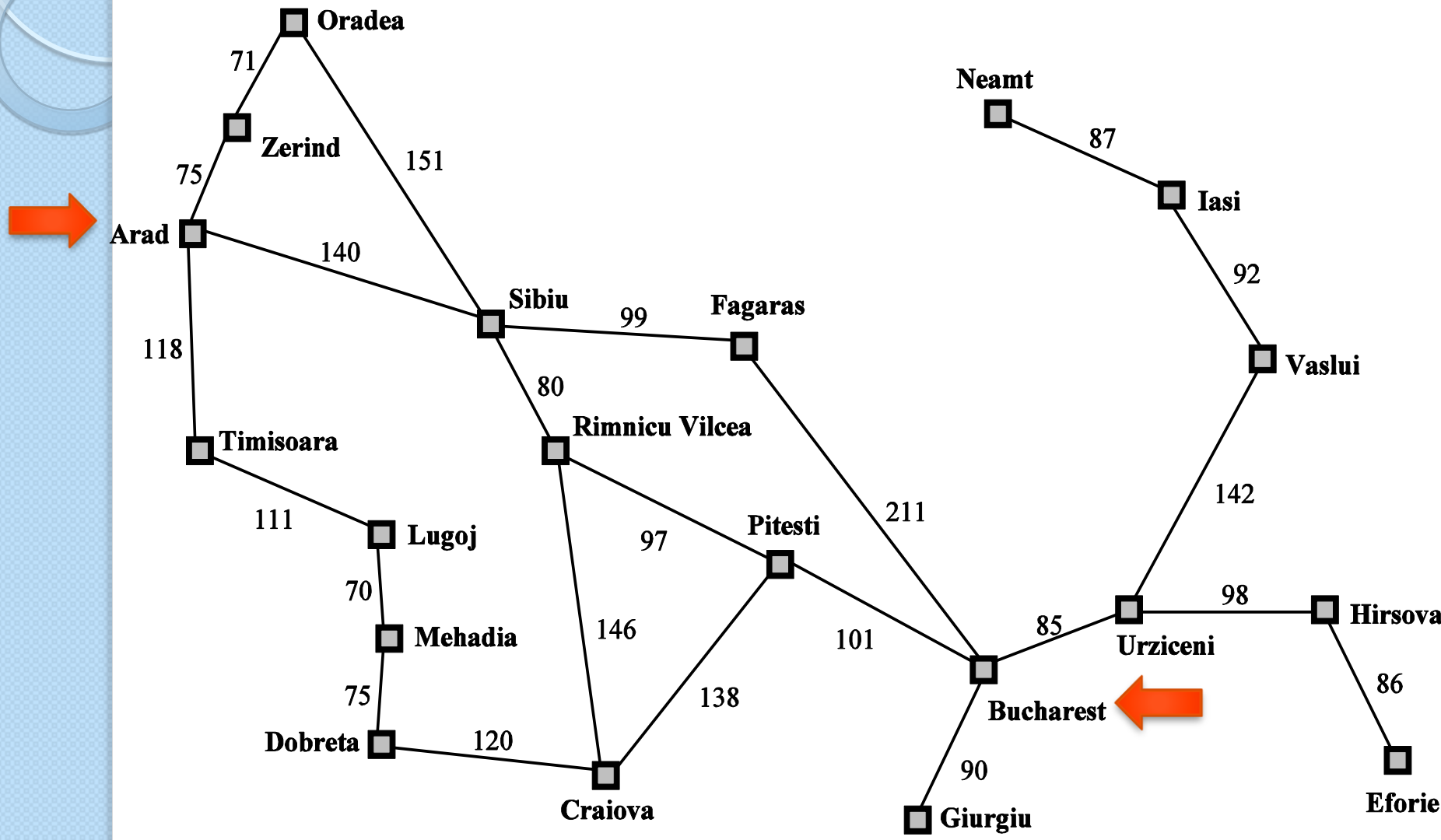
Thuật toán		Đủ	Tối ưu	Thời gian	Không gian
BestFS	Best First Search	Y	**N**	$O(min(N,B^{LMAX}))$	$O(min(N,B^{LMAX}))$

GREEDY BEST FIRST SEARCH



- Greedy Best First Search không đảm bảo tìm thấy đường đi tối ưu.
- Câu hỏi đặt ra: *Làm thế nào khắc phục điều này?*

BÀI TẬP VÍ DỤ

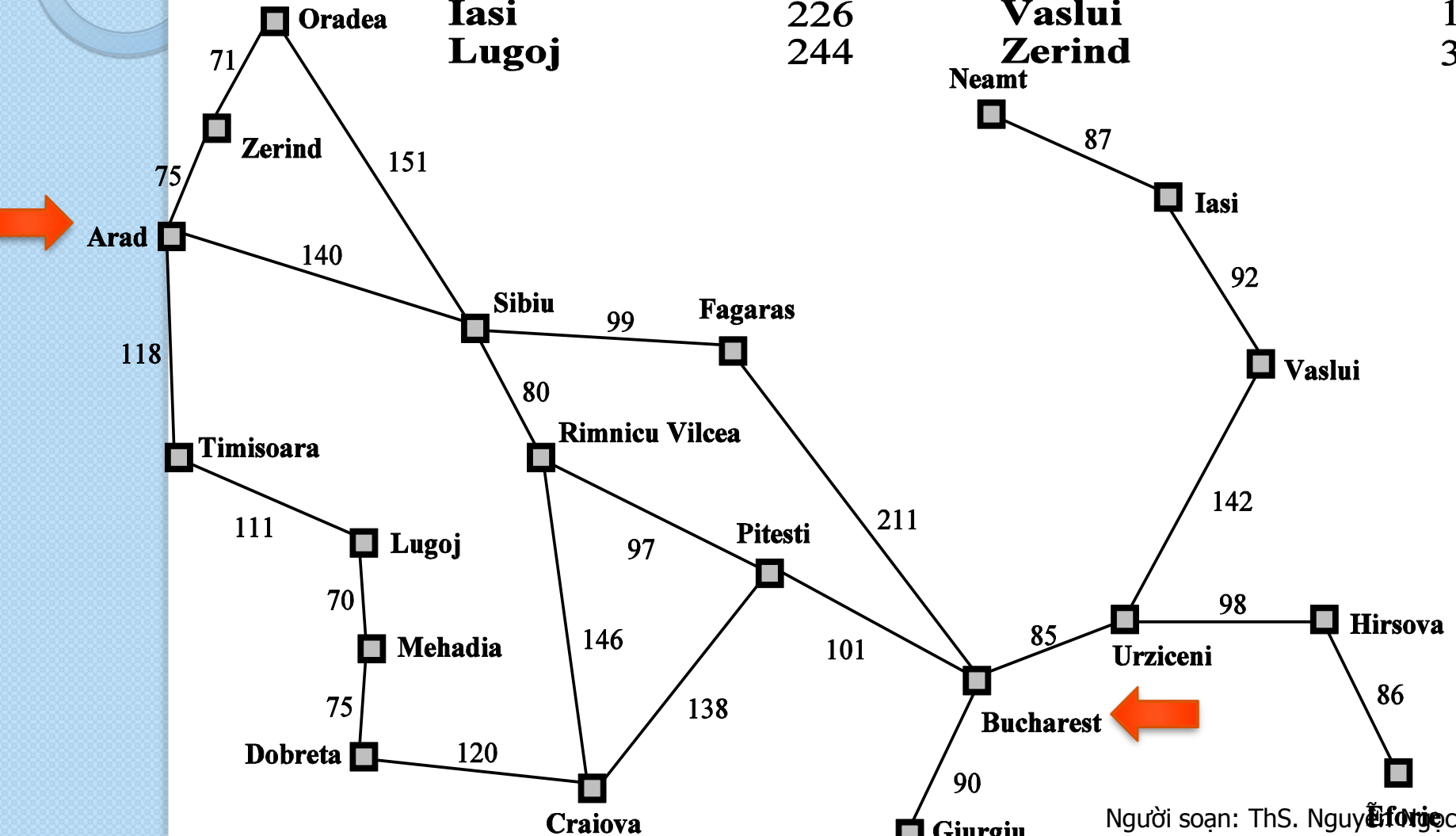

Oradea
71
Zerind
75
151
Arad
140
Sibiu
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
75
138
Dobreta
120
Craiova
90
Giurgiu
Neamt
87
Iasi
92
Vaslui
142
98
Hirsova
85
Urziceni
86
Bucharest
Eforie

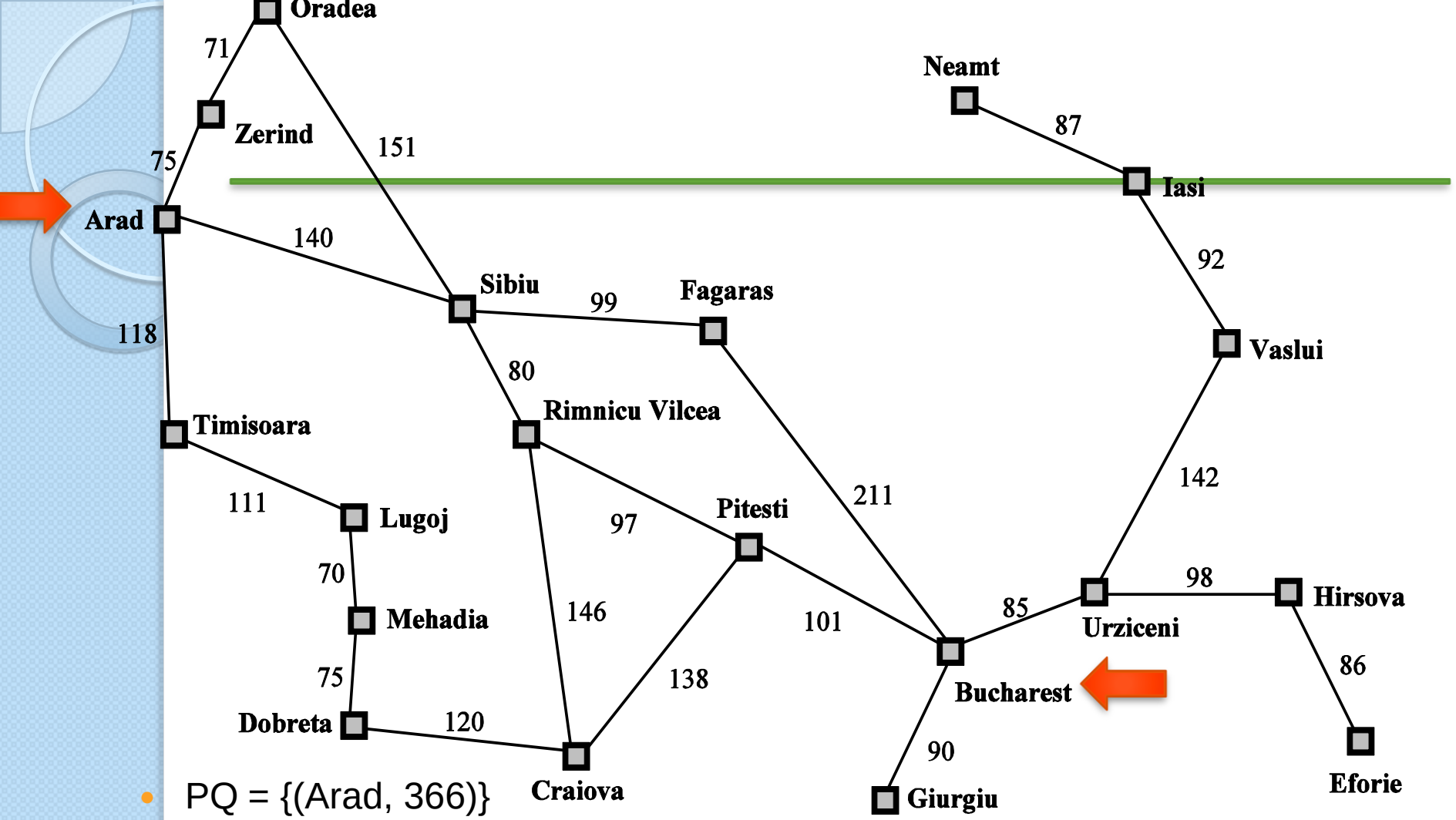
BÀI TẬP VÍ DỤ

- Cho trước heuristic *h(x)* là khoảng cách đường chim bay giữa các thành phố (bảng bên dưới)

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374





- PQ = {(Arad, 366)}
- PQ = {(Sibiu, 253), (Timisoara, 329), (Zerind,374)}
- PQ = {(Faragas, 176), (Rimnicu Vilcea, 193), (Timisoara, 329), (Zerind,374)}
- PQ = {(Bucharest, 0), (Rimnicu Vilcea, 193), (Timisoara, 329), (Zerind,374)}

Đường đi là: Arad → Sibiu → Faragas → Bucharest Chi phí: 450

TÌM KIẾM A*

(A* Search)

Ý TƯỞNG CƠ BẢN

- Greedy Best First Search: Khi mở nút *s,* xác định mỗi nút con *s'* và đặt vào PriQueue với độ ưu tiên *h*(s').
- A*: Khi mở nút *s*, xác định mỗi nút con *s'* và đặt vào PriQueue với độ ưu tiên

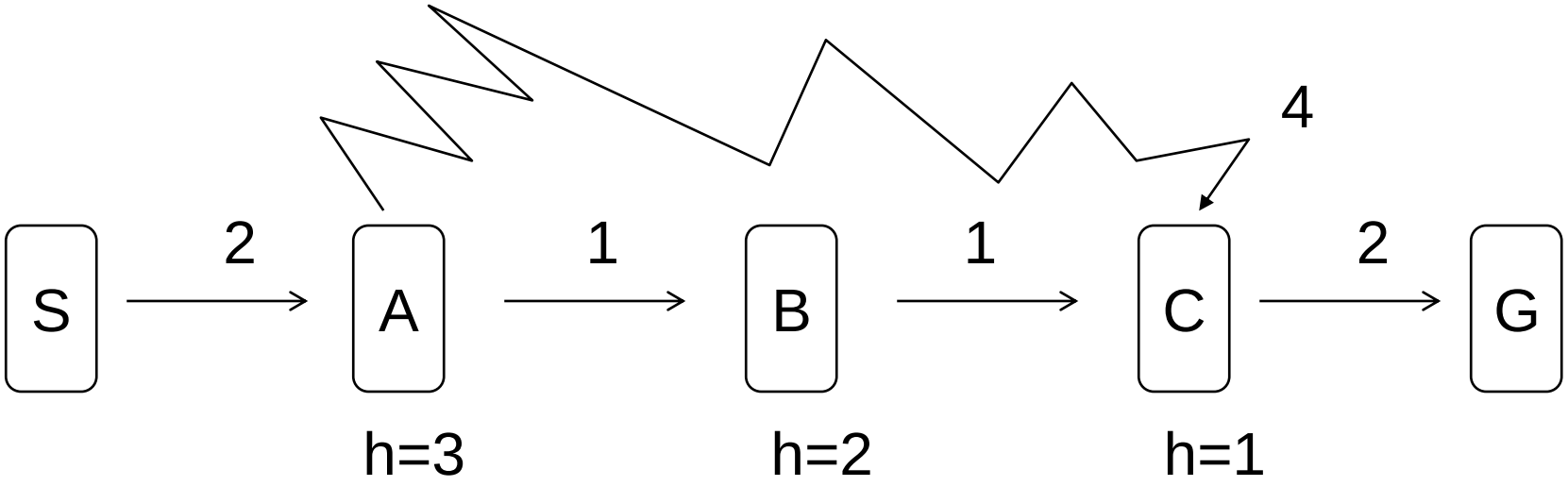
  (Chi phí đi đến s') + h(s') (1)

  Gọi g(s) = Chi phí đi đến n (2)

  và định nghĩa

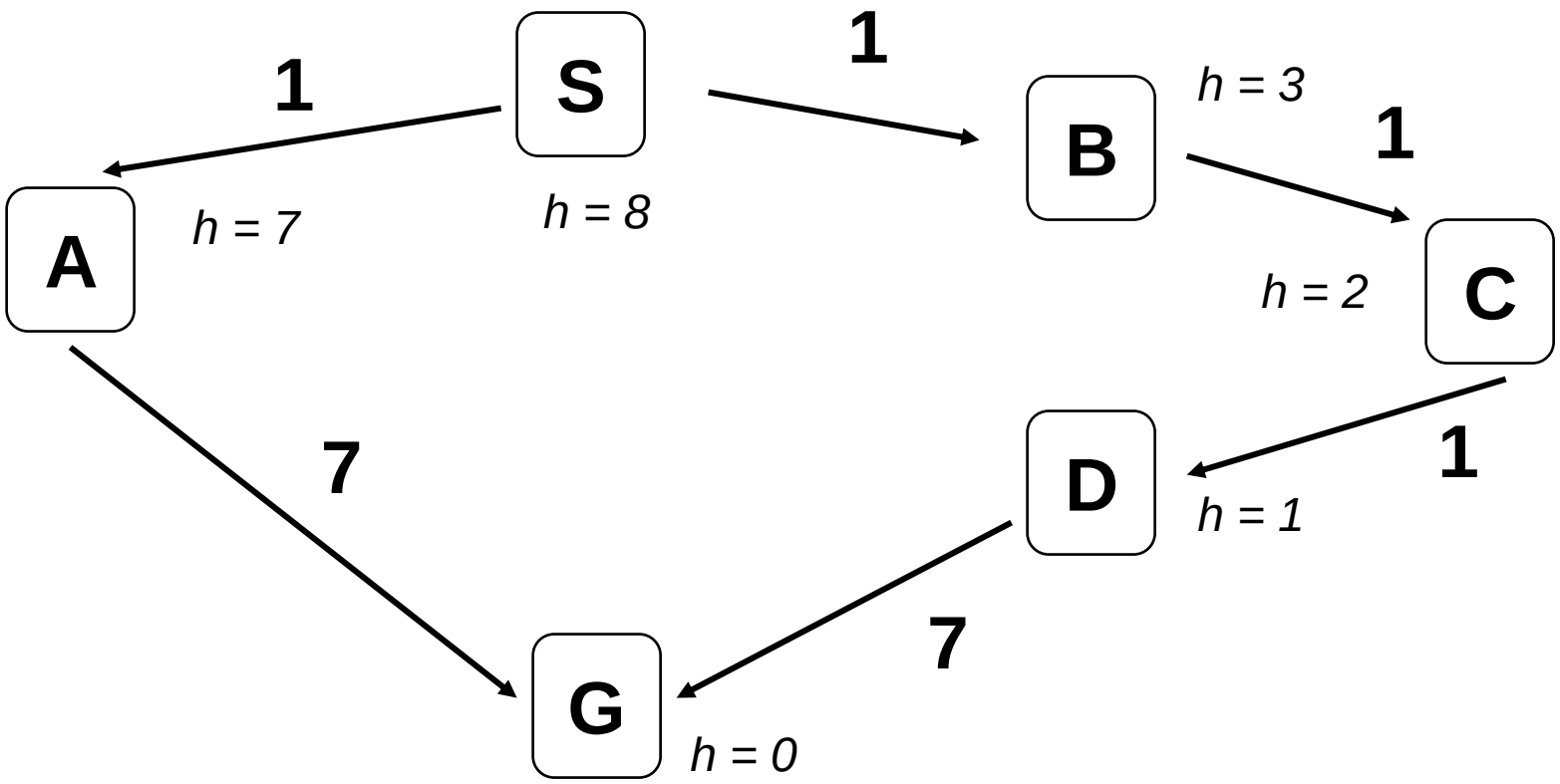
  f(s) = g(s) + h(s) (3)

Ý TƯỞNG CƠ BẢN



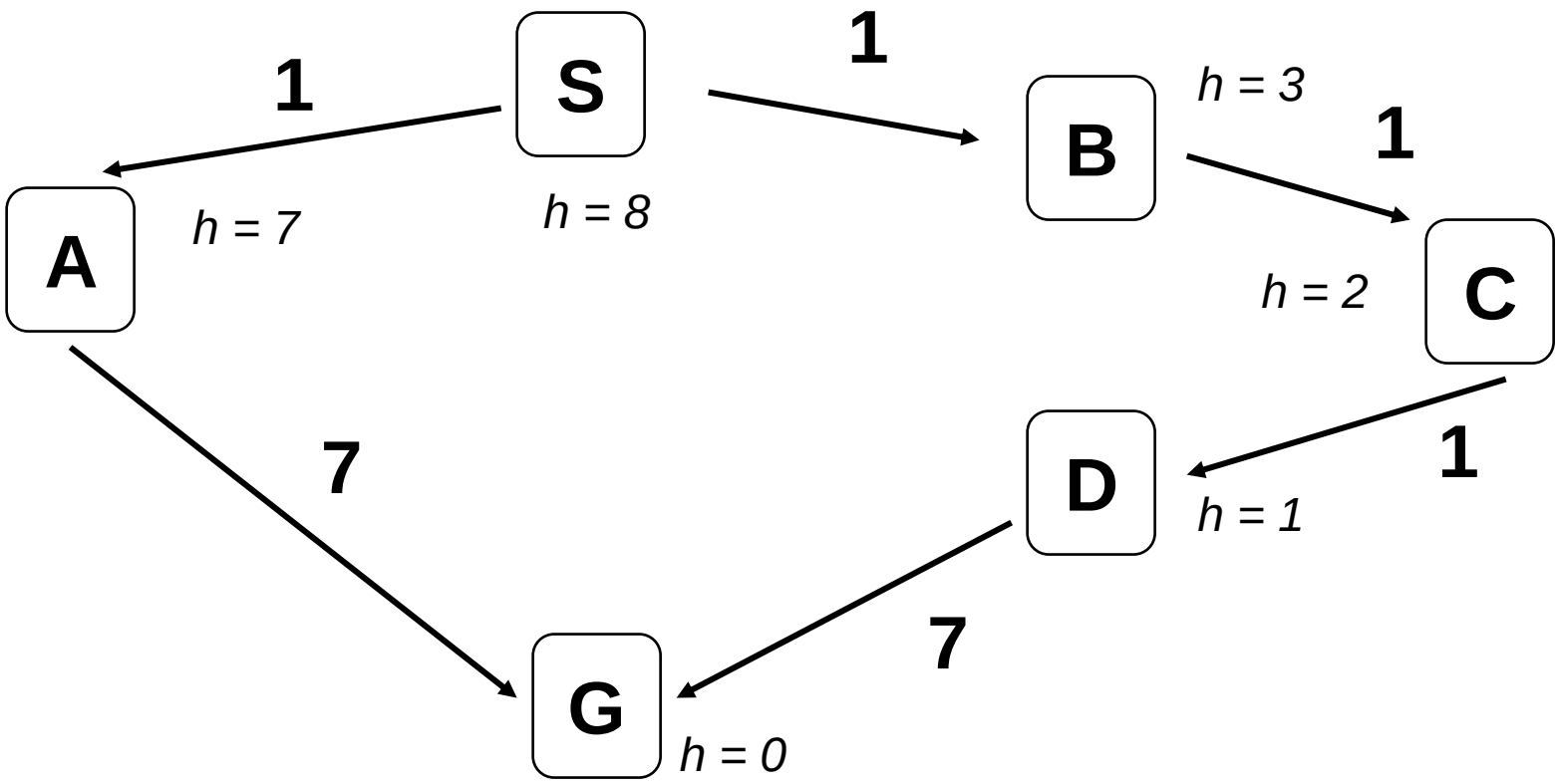
ĐIỀU KIỆN DỪNG

- Ý tưởng: Ngay khi A* phát sinh ra trạng thái đích?
- Hãy quan sát ví dụ sau:



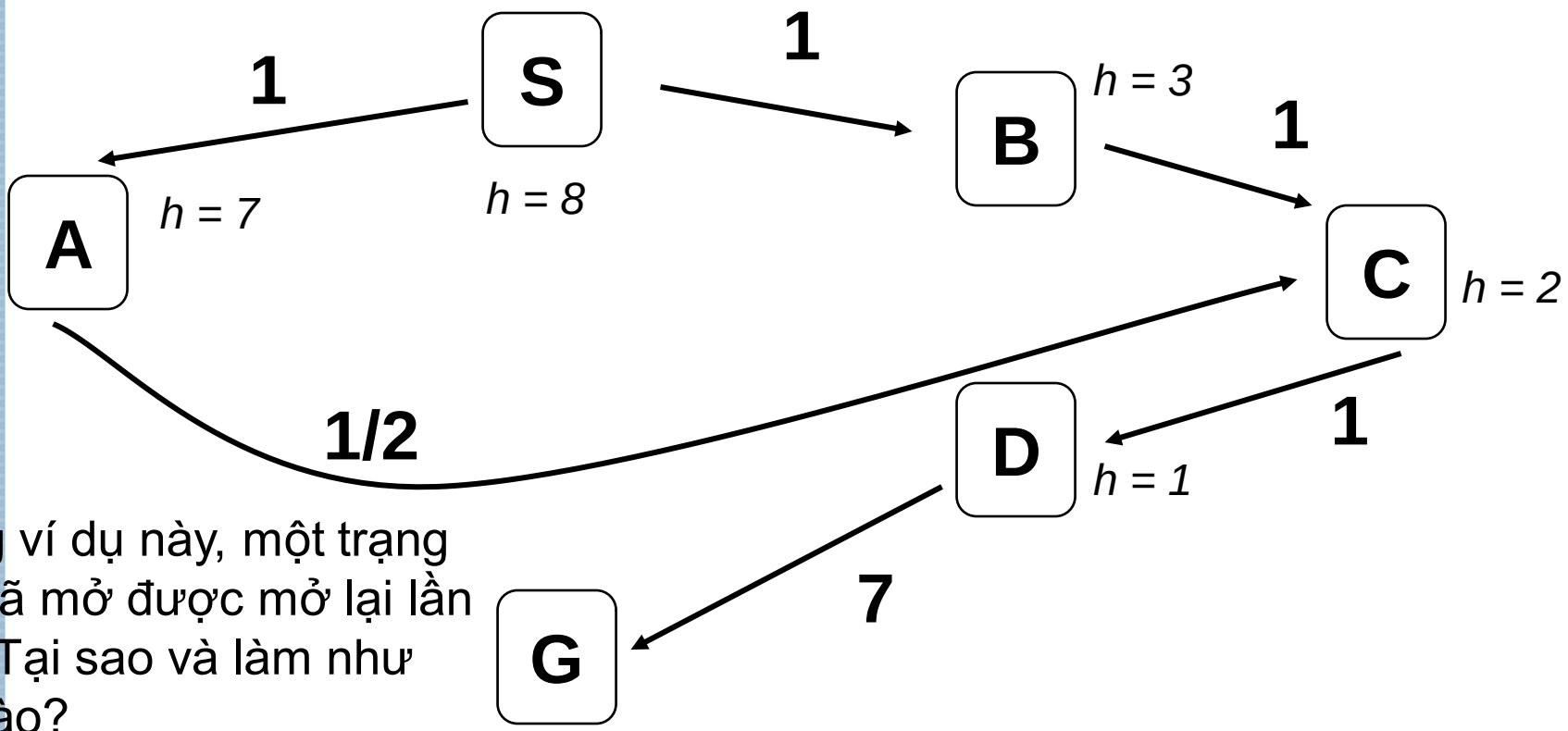
ĐIỀU KIỆN DỪNG

- A* chỉ dừng khi trạng thái đích được lấy ra khỏi hàng đợi ưu tiên.



A* – DUYỆT LẠI TRẠNG THÁI

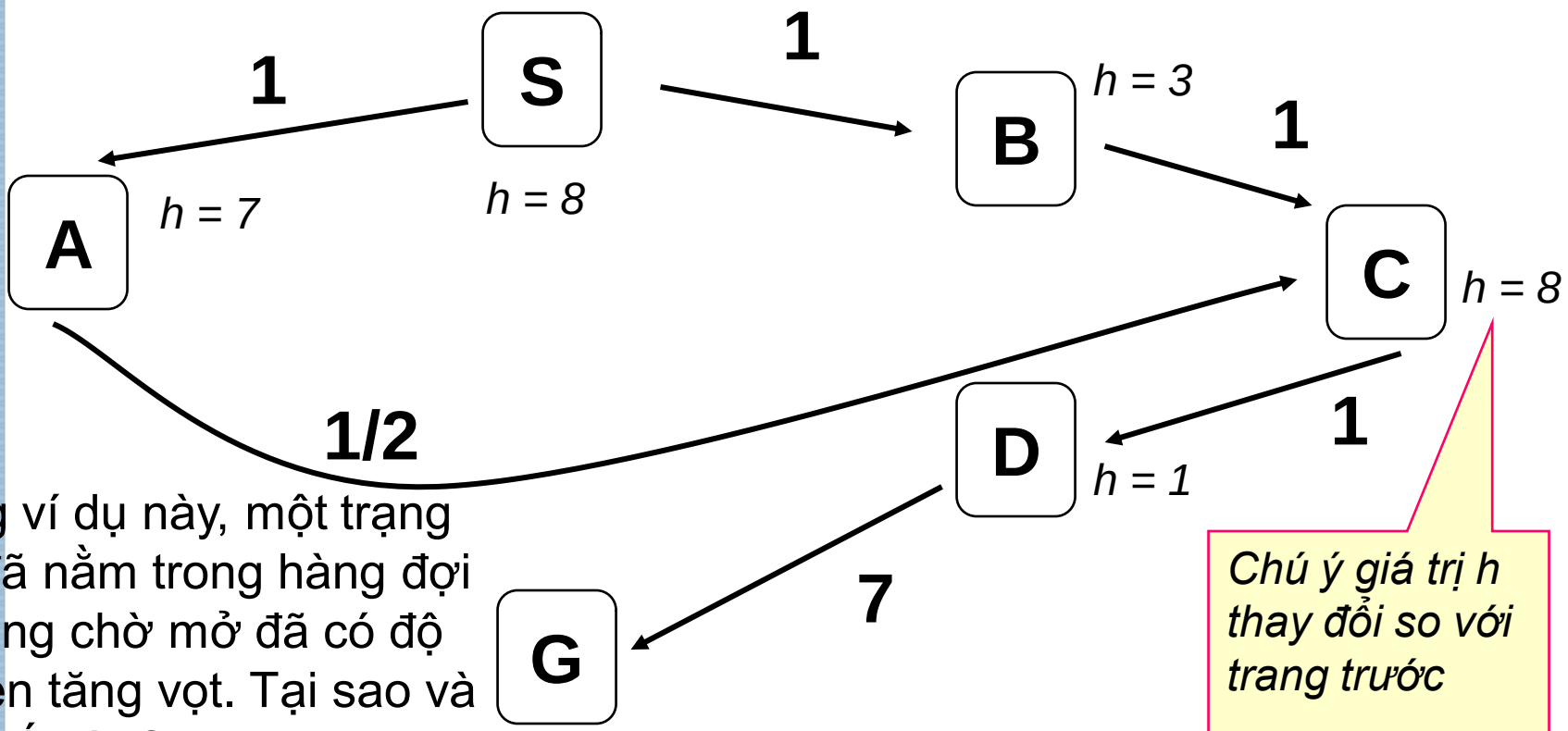
- Điều gì xảy ra nếu A* duyệt lại một trạng thái đã mở và tìm thấy đường đi khác ngắn hơn?



Trong ví dụ này, một trạng thái đã mở được mở lại lần nữa. Tại sao và làm như thế nào?

A* – DUYỆT LẠI TRẠNG THÁI

- Điều gì xảy ra nếu A* duyệt một trạng thái đang nằm trong hàng đợi?



Trong ví dụ này, một trạng thái đã nằm trong hàng đợi và đang chờ mở đã có độ ưu tiên tăng vọt. Tại sao và làm thế nào?

Chú ý giá trị h thay đổi so với trang trước

THUẬT TOÁN A*

- Init-PriQueue(PQ)
- V = {} // V = { (s, f, con trỏ quay lui) } - tập trang thái đi qua.
- PQ ← S với f(s) = g(s) + h(s) // đưa s vào PQ
- while PQ != rỗng
  - s ← PQ // lấy s với f thấp nhất ra
  - If s == g then → dừng và thông báo thành công
  - Với mỗi s' trong **succs**(*s*)…
    - f’ = g(s') + h(s') // = g(s) + cost(n, s') + h(s')
    - **If** s' chưa đi qua

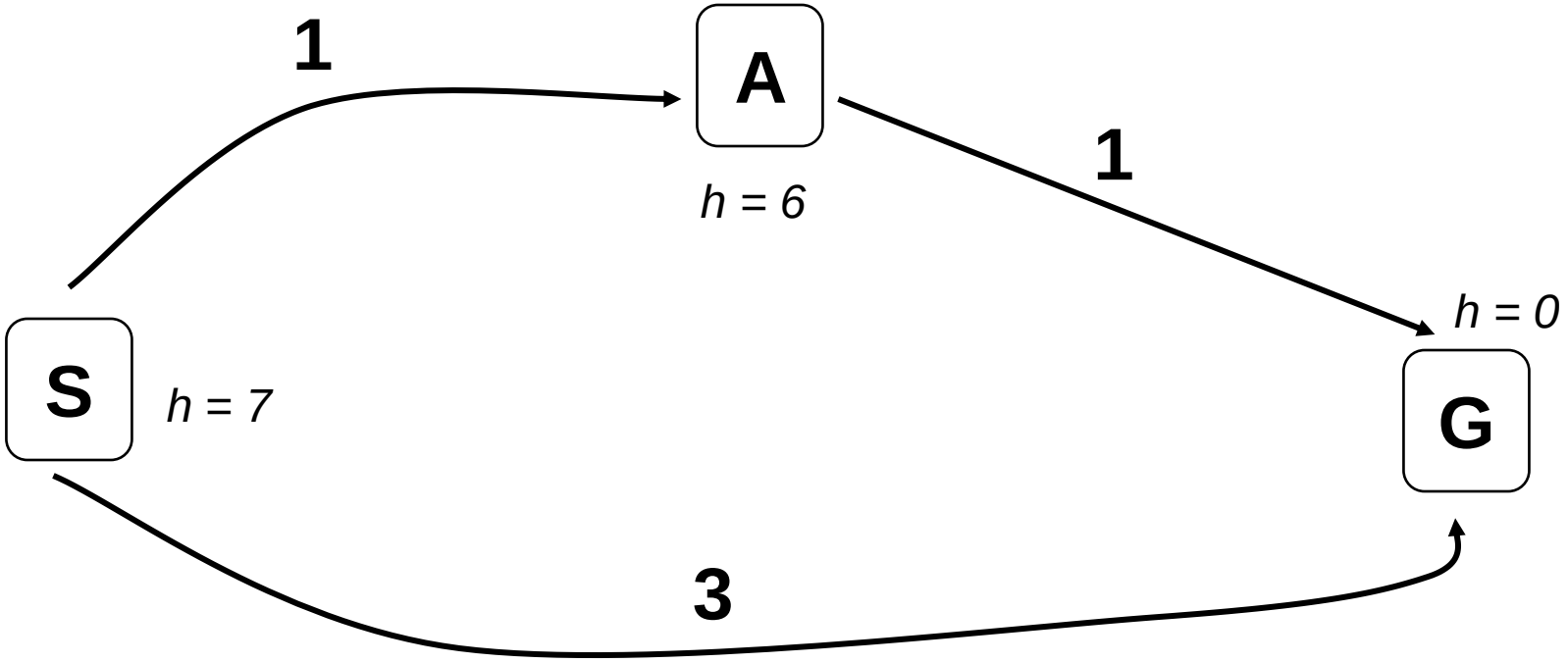
      PQ ← s‘ //thêm hoặc câp nhật f’ của s’ mới nhỏ hơn

      else

      Cập nhật f’ của s’ trong V nếu nhỏ hơn s’ cũ

    V ← s

A* CÓ BẢO ĐẢM TỐI ƯU?



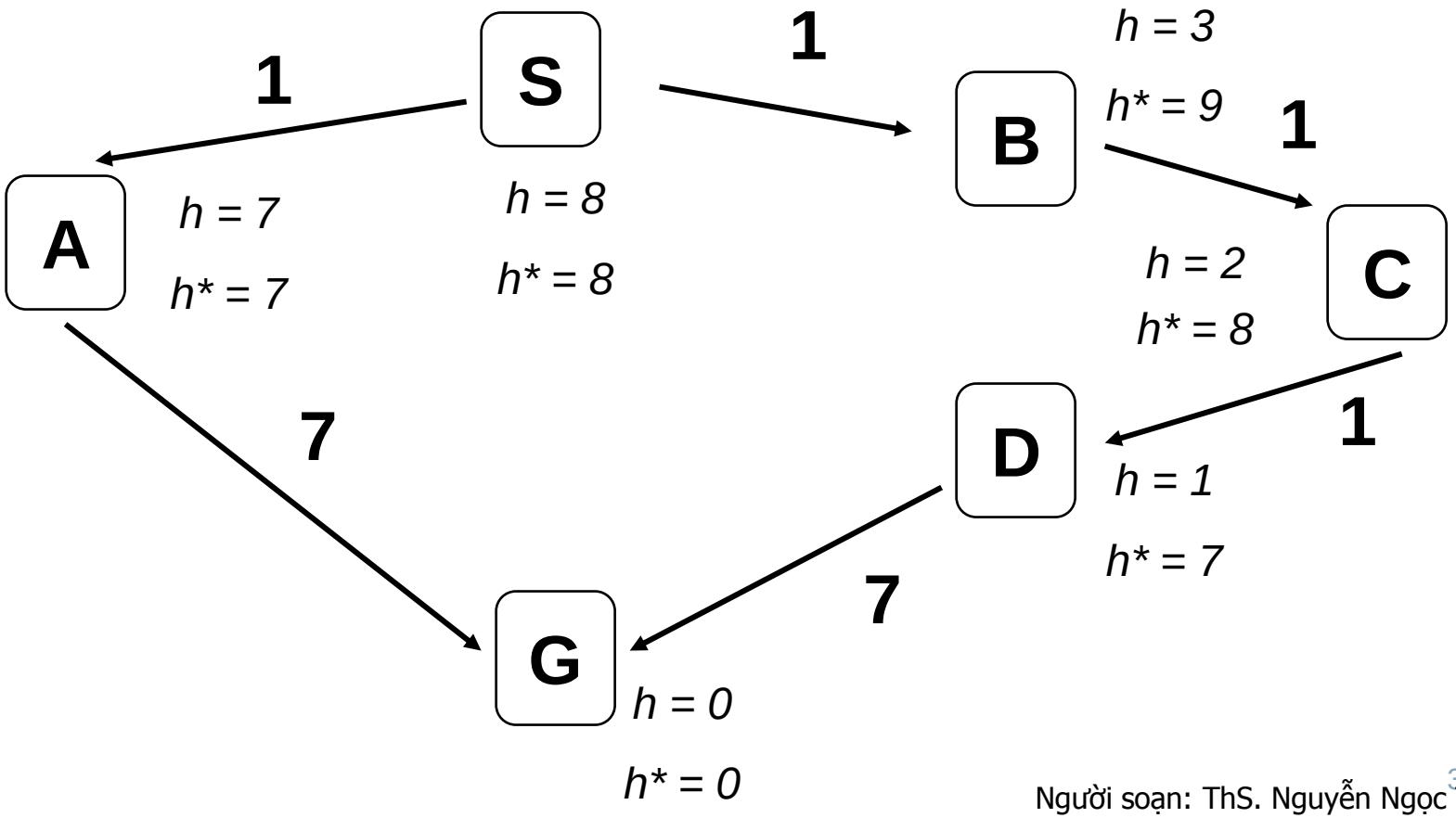
- Không. Hãy quan sát ví dụ trên.

HEURISTIC CHẤP NHẬN ĐƯỢC

- Gọi $h^*(s)$ = chi phí tối thiểu thật sự từ trạng thái s đến đích.
- Một heuristic h là chấp nhận được nếu
  $h(s) \leq h^*(s)$ với mọi trạng thái s
- Heuristic chấp nhận được đảm bảo không bao giờ ước tính quá chi phí đến đích.
- Heuristic chấp nhận được là tối ưu.

h*

- *h**(s) = chi phí **tối thiểu thật sự** từ trạng thái s đến đích.



VÍ DỤ 8-PUZZLE

START

1		5
2	6	3
7	4	8

GOAL

1	2	3
4	5	6
7	8	

Heuristic nào sau đây là chấp nhận được?

- *h*(s) = Số ô nằm sai vị trí trong trạng thái s
- *h*(s) = 0
- *h*(s) = Tổng khoảng cách Manhattan giữa từng ô và vị trí đích
- *h*(s) = 1
- *h*(s) = min (2, *h**[s])
- *h*(s) = *h*(s)*
- *h*(s) = max (2, *h**[s])

A* VÀ LẶP SÂU DẦN

	Với 8-puzzle, số trạng thái được mở trung bình trong 100 bài toán được chọn ngẫu nhiên trong đó đường đi tối ưu dài…		
	…4 bước	…8 bước	…12 bước
Lặp sâu dần	112	6,300	3.6×10^6
A* với heuristic "số ô sai vị trí"	13	39	227
A* với heuristic "tổng khoảng cách Manhattan"	12	25	73

Page 104, chapter 3 - S. Russel and P.Norvig, Artificial Intelligence – A Modern Approach. *Third Edition*. 2010.

A* VÀ [illegible]

Thật sự chỉ có một vài trăm ngàn trạng thái cho toàn bộ bài toán 8-puzzle.

Ghi chú:

1. Thoạt nhìn ta thấy "số ô sai" có thể là một heuristic tốt. Nhưng ngay cả h(state) = 0 cũng có thể làm tốt hơn ID, vậy điểm khác biệt chủ yếu ở đây là vấn đề nghiêm trọng của ID khi mở cùng một trạng thái nhiều lần, không phải do dùng heuristic.
2. Đánh giá chỉ trên "số trạng thái đã mở" không tính đến chi phí quá mức để duy trì bảng băm và hàng đợi ưu tiên cho A*, mặc dù ta thấy khá rõ là điều đó không làm thay đổi đáng kể kết quả.

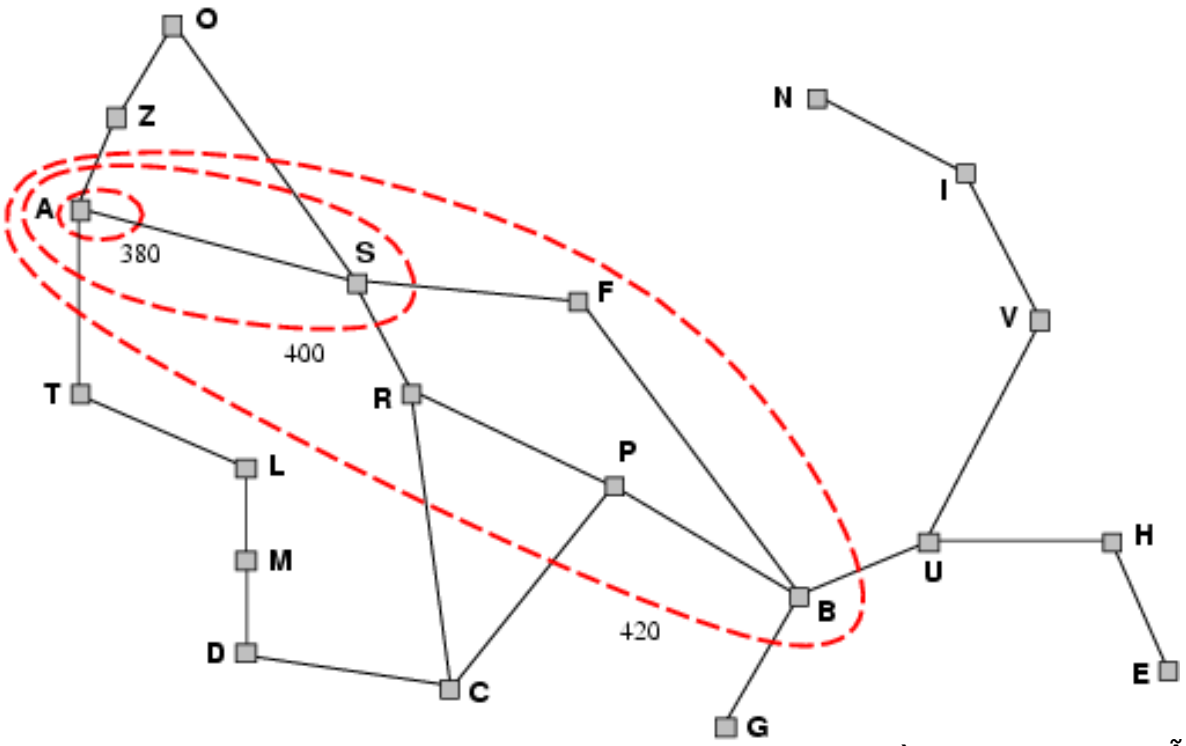
[illegible]g thái được
[illegible]100 bài toán
[illegible]ên trong đó
[illegible].

			…12 bước
		[illegible]9,300	3.6 x 10^6
A* với heuristic "số ô sai vị trí"	13	39	227
A* với heuristic "tổng khoảng cách Manhattan"	12	25	73

Page 104, chapter 3 - S. Russel and P.Norvig, Artificial Intelligence – A Modern Approach. *Third Edition*. 2010.

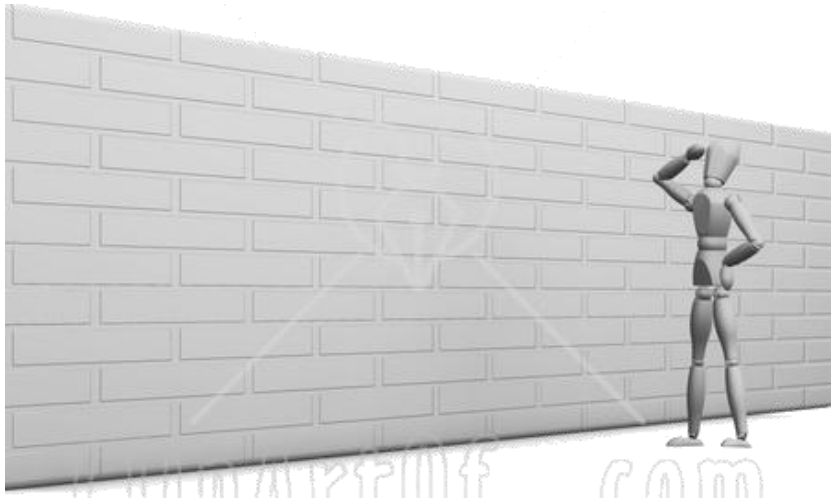
ƯU ĐIỂM A*

- A* mở các nút để tổng hoảng cách từ nút đó tới Start nhỏ nhưng vẫn hướng tới đích.



CÁC NHƯỢC ĐIỂM

- A* có thể dùng nhiều bộ nhớ. Về lý thuyết, A* có độ phức tạp là *O*(số trạng thái).
- Với các không gian tìm kiếm lớn, A* sẽ tràn bộ nhớ.

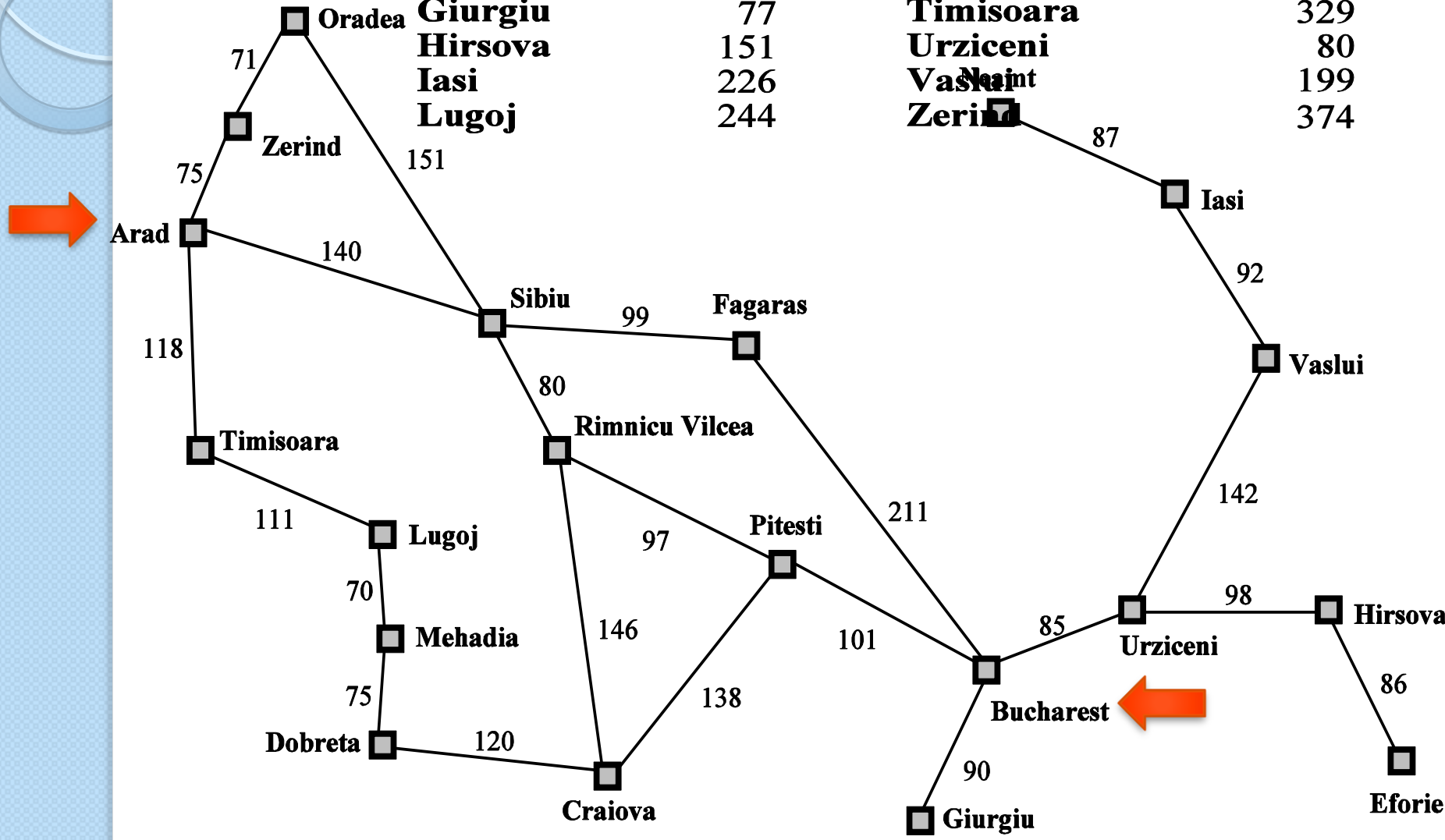


BÀI TẬP VÍ DỤ

- Cho trước heuristic *h(x)* là khoảng cách đường chim bay giữa các thành phố (bảng bên dưới)

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



BÀI TẬP VÍ DỤ

- PQ = {(Arad, 366)}
- PQ = {(Sibiu, 393), (Timisoara, 447), (Zerind,449)}
- PQ = {(Rimnicu Vilcea, 413), (Faragas, 415), (Timisoara, 447), (Zerind,449)}
- PQ = {(Faragas, 415), (Pitesti, 417), (Craiova, 526), (Timisoara, 447), (Zerind,449)}
- PQ = {(Pitesti, 417), (Craiova, 526), (Timisoara, 447), (Zerind,449), (Bucharest, 450)}
- PQ = {(Bucharest, 418), (Craiova, 526), (Timisoara, 447), (Zerind,449)}

Đường đi là: Arad → Sibiu → Rimnucu Vilcea → Pitesti → Bucharest

THUẬT GIẢI LEO ĐỒI

(Hill-Climbing Search)

THUẬT GIẢI LEO ĐỒI

- Các thuật toán tìm kiếm toàn cục: sử dụng quá nhiều tài nguyên (A*) hoặc thời gian (BFS) để tìm được lời giải tối ưu.

  Ta có thể thực hiện việc tìm kiếm lời giải trong thời gian và không gian hợp lý?

THUẬT GIẢI LEO ĐỒI

- Cố gắng tối đa hóa Eval(X) bằng cách di chuyển đến cấu hình cao nhất trong tập di chuyển của mình – Leo đồi dốc đứng

Đặt S := trạng thái ban đầu

Lặp

Tìm trạng thái con S' của S với Eval(S') tốt nhất

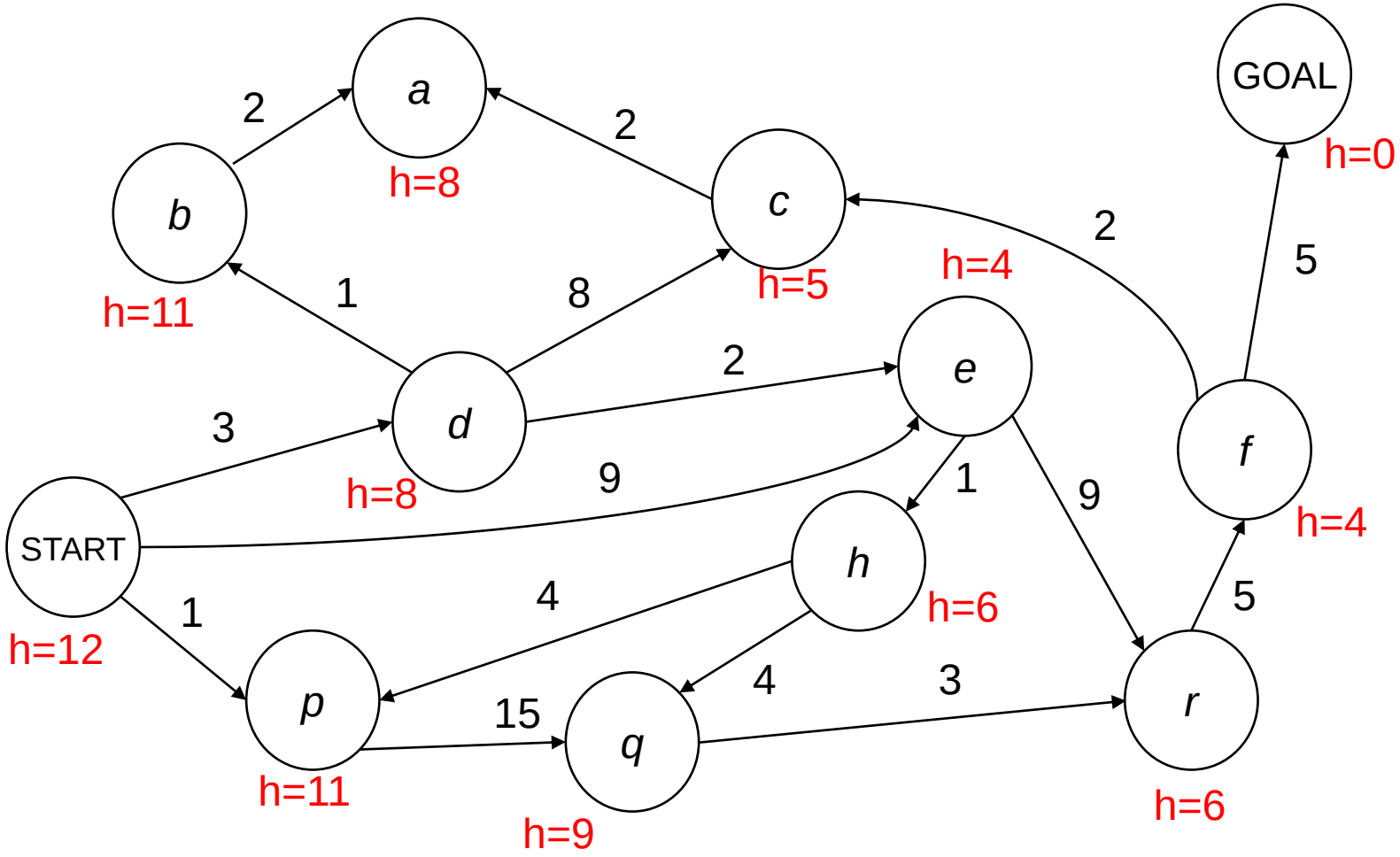
Nếu Eval(S') không tốt hơn Eval(S) thì

return S

Ngược lại

S = S'

THUẬT GIẢI LEO ĐỒI



LEO ĐỒI NGẪU NHIÊN

Đặt S := trạng thái ban đầu
Lặp sau một MAX lần cố gắng nào đó
Lấy một trạng thái con ngẫu nhiên S' của S
Nếu Eval(S') tốt hơn Eval(S) thì
S= S'
Cuối lặp
Return S

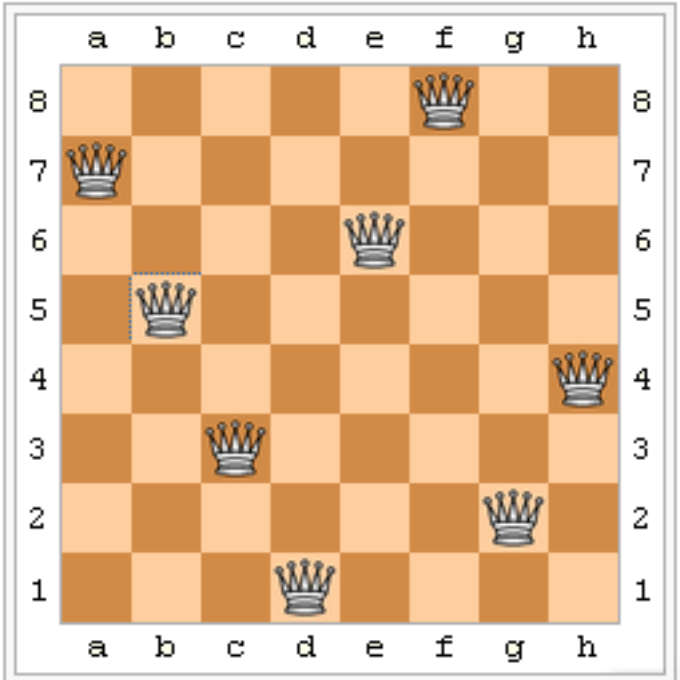
Sau khi chạy vài lần có thể đưa đến trạng thái đích

BÀI TOÁN TỐI ƯU HÓA

- Ta chỉ quan tâm đến việc đạt được một cấu hình tối ưu mà không cần quan tâm đến đường đi.
- Xây dựng một tập di chuyển (moveset) từ một trạng thái sang một trạng thái khác
- Phát sinh ngẫu nhiên trạng thái ban đầu
- Thực hiện di chuyển xuống (lên) đồi

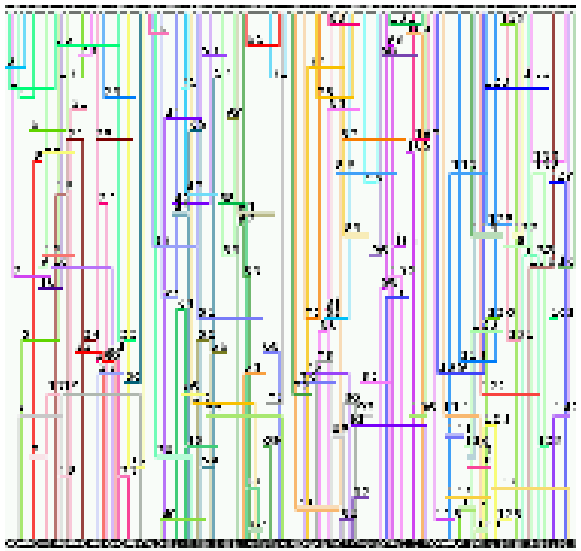
VÍ DỤ BÀI TOÁN TỐI ƯU HÓA

- Bài toán n-Hậu
  - Đây là một bài toán Thỏa mãn Ràng buộc (Contraint Satisfaction Problem – CSP)
  - Có thể xem xét dưới dạng một bài toán tối ưu hoá với hàm lượng giá ***h*** = *số lượng cặp hậu đe doạ lẫn nhau*

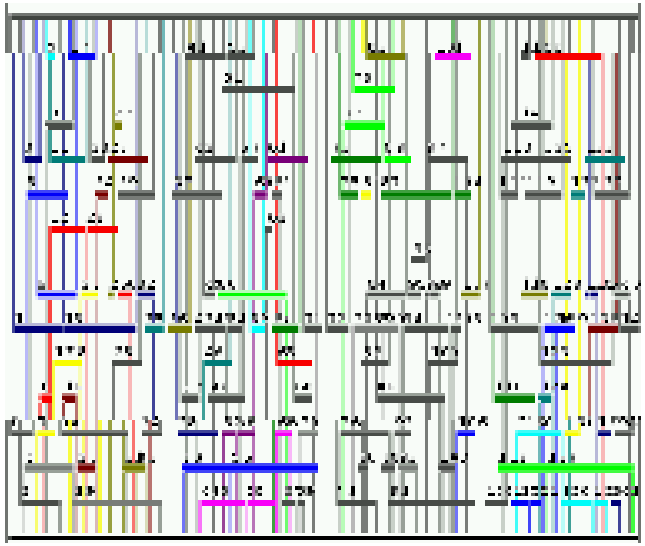


One possible solution

VÍ DỤ BÀI TOÁN TỐI ƯU HÓA



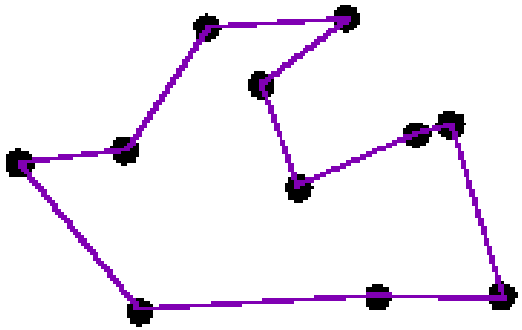
Thiết kế
Mạch điện



Có rất nhiều chip cố định

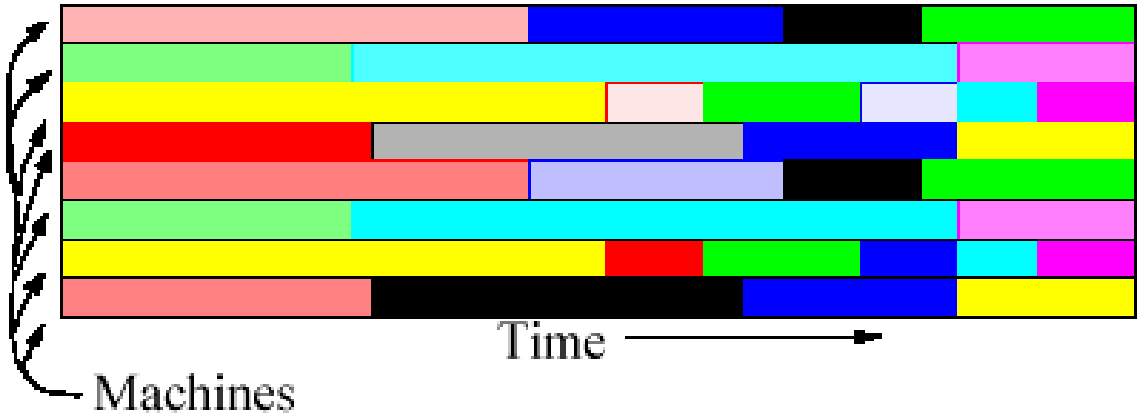
Cùng số kết nối
nhưng tốn ít
không gian hơn

Travelling
Salesperson
Problem



VÍ DỤ BÀI TOÁN TỐI ƯU HÓA

Least cost, constrained, schedule



Boolean SATisfiability

$$A \vee \neg B \vee C$$

$$\neg A \vee C \vee D$$

$$B \vee D \vee \neg E$$

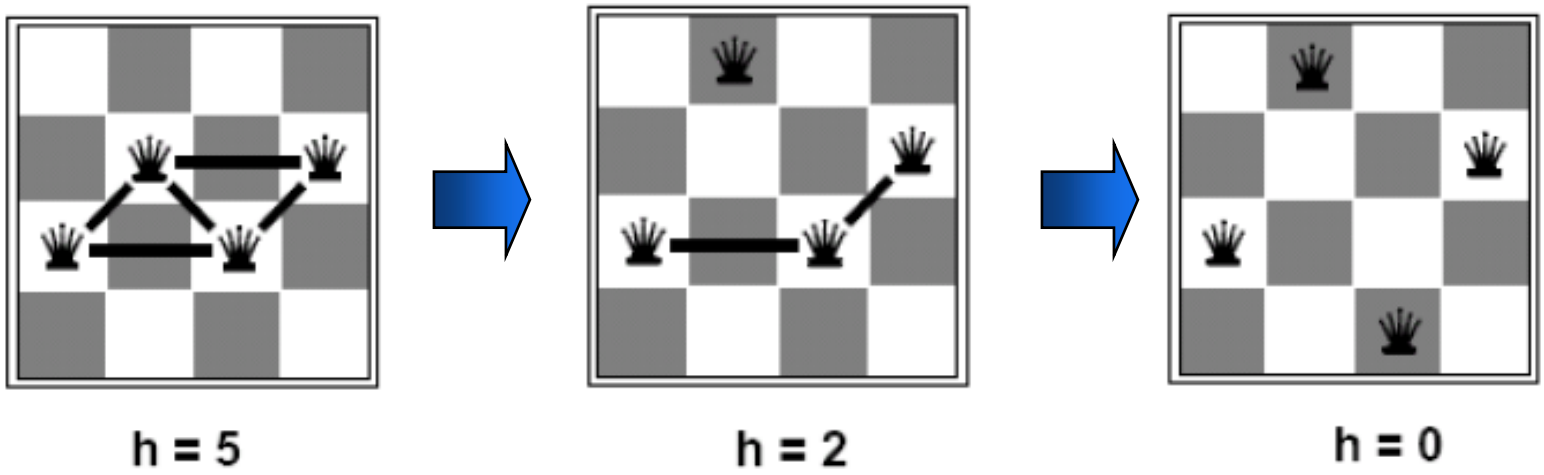
$$\neg C \vee \neg D \vee \neg E$$

$$\neg A \vee \neg C \vee E$$

(2000 variables, 8500 clauses)

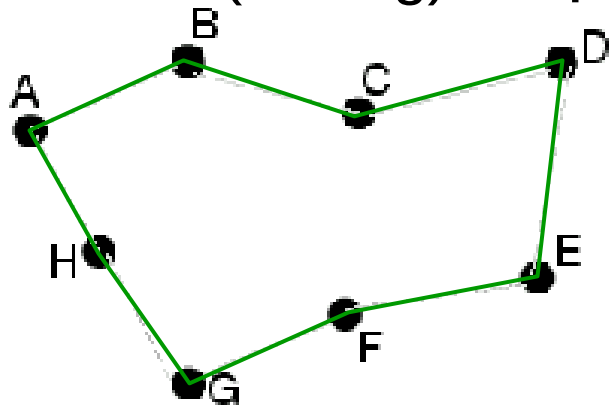
VÍ DỤ BÀI TOÁN TỐI ƯU HÓA

- Thuật giải leo đồi thực hiện với bài toán n-Hậu

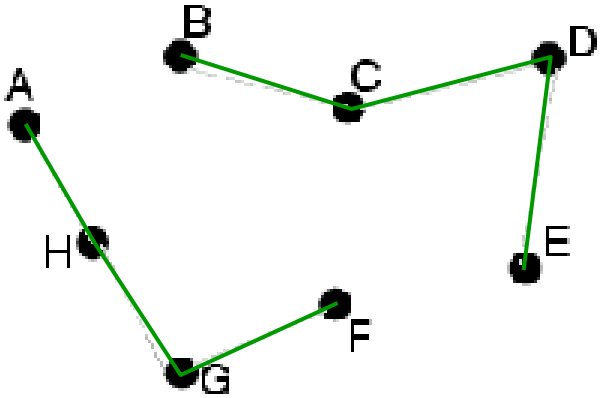


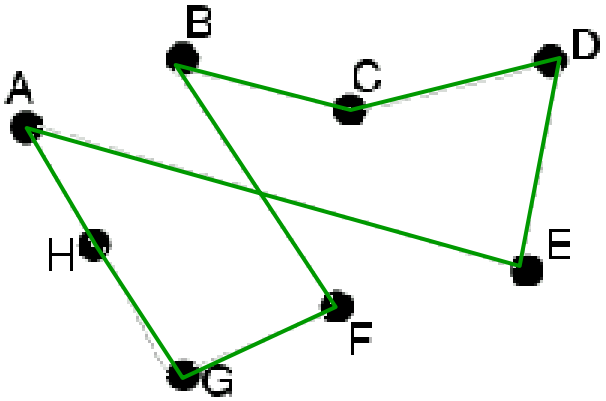
VÍ DỤ BÀI TOÁN TỐI ƯU HÓA

- Tối thiểu hóa: Eval(Config) = độ dài đường đi



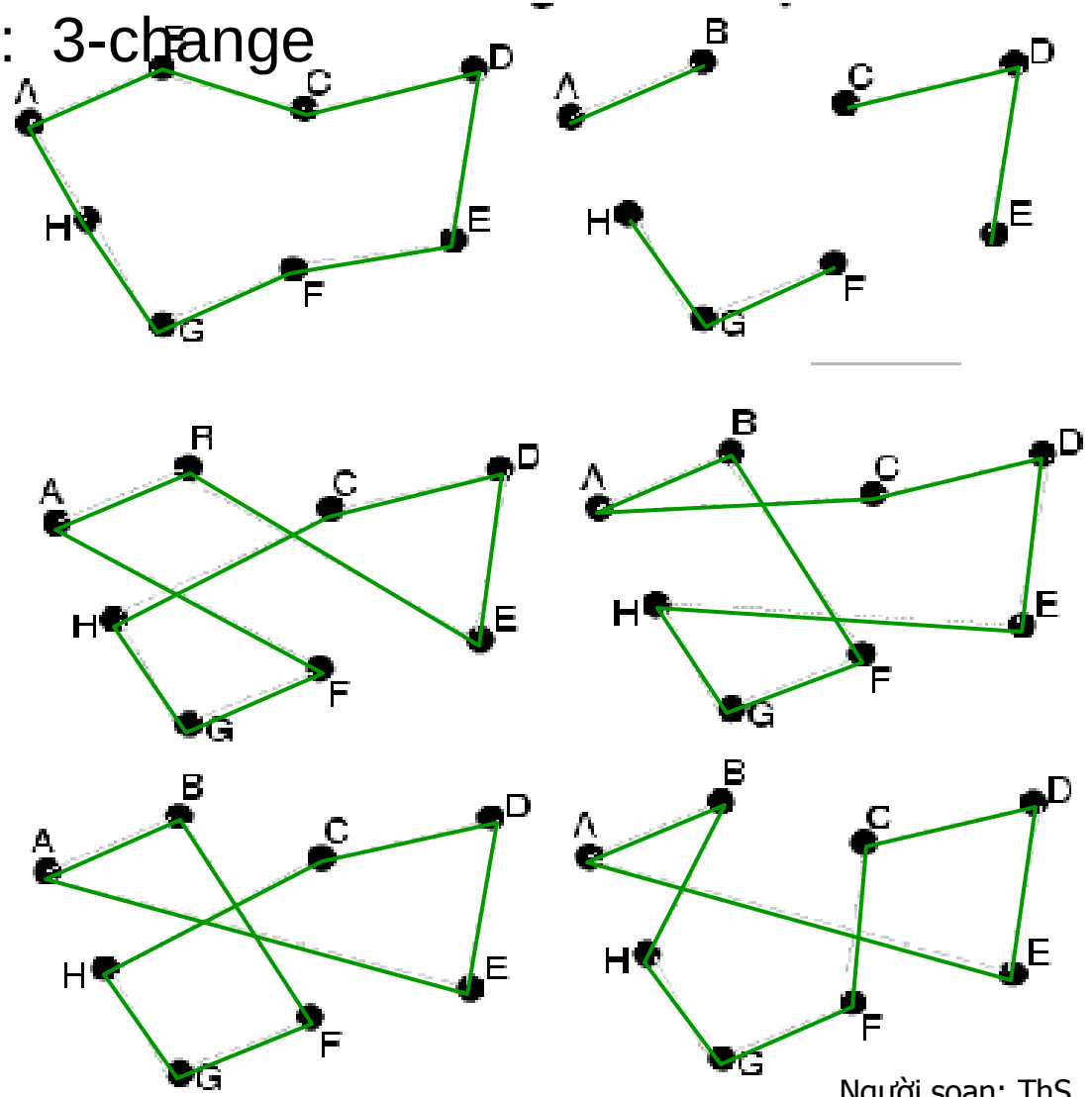
- Tập di chuyển: 2-change … k-change.
- Ví dụ: 2-change



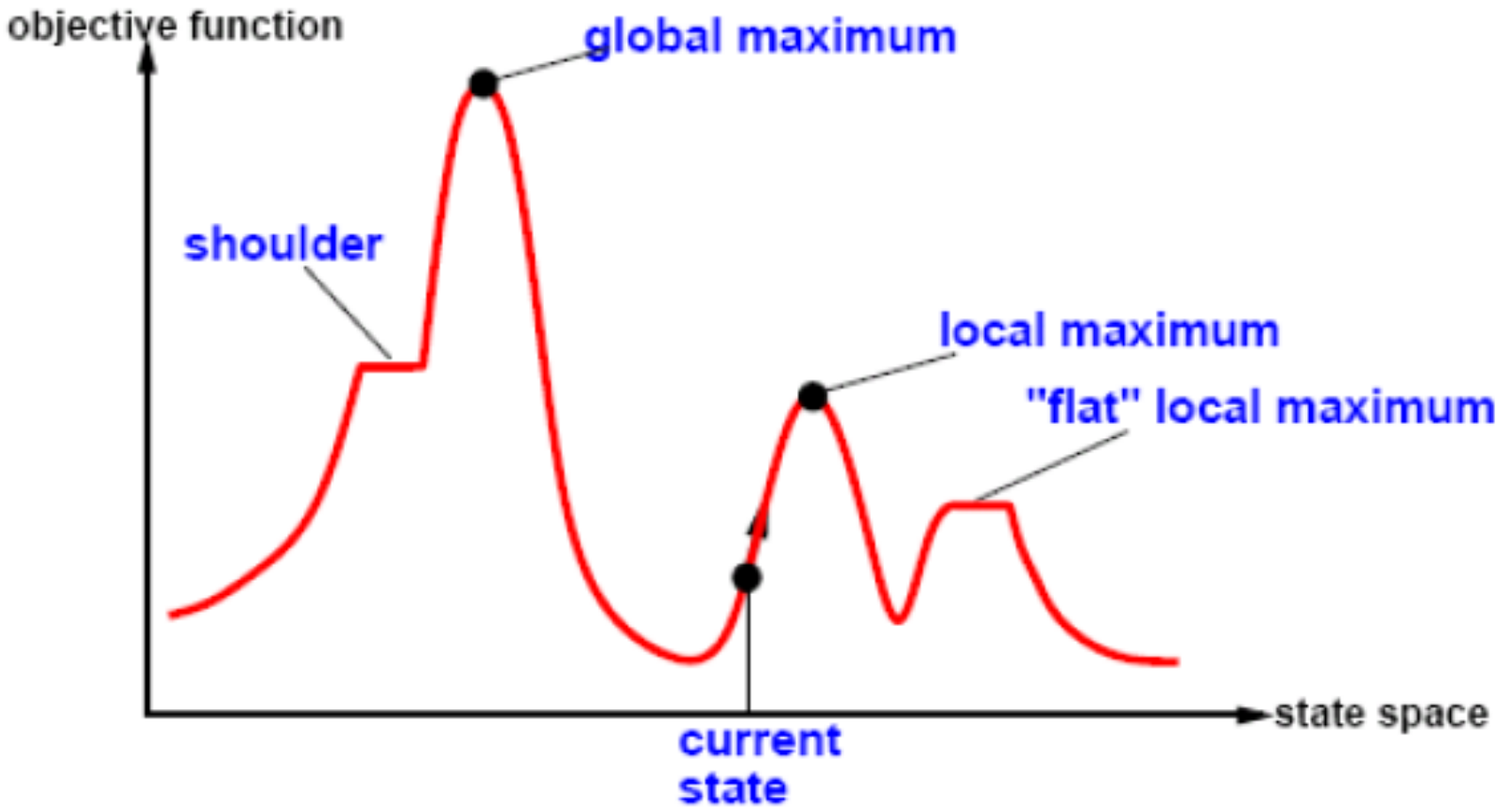


VÍ DỤ BÀI TOÁN TỐI ƯU HÓA

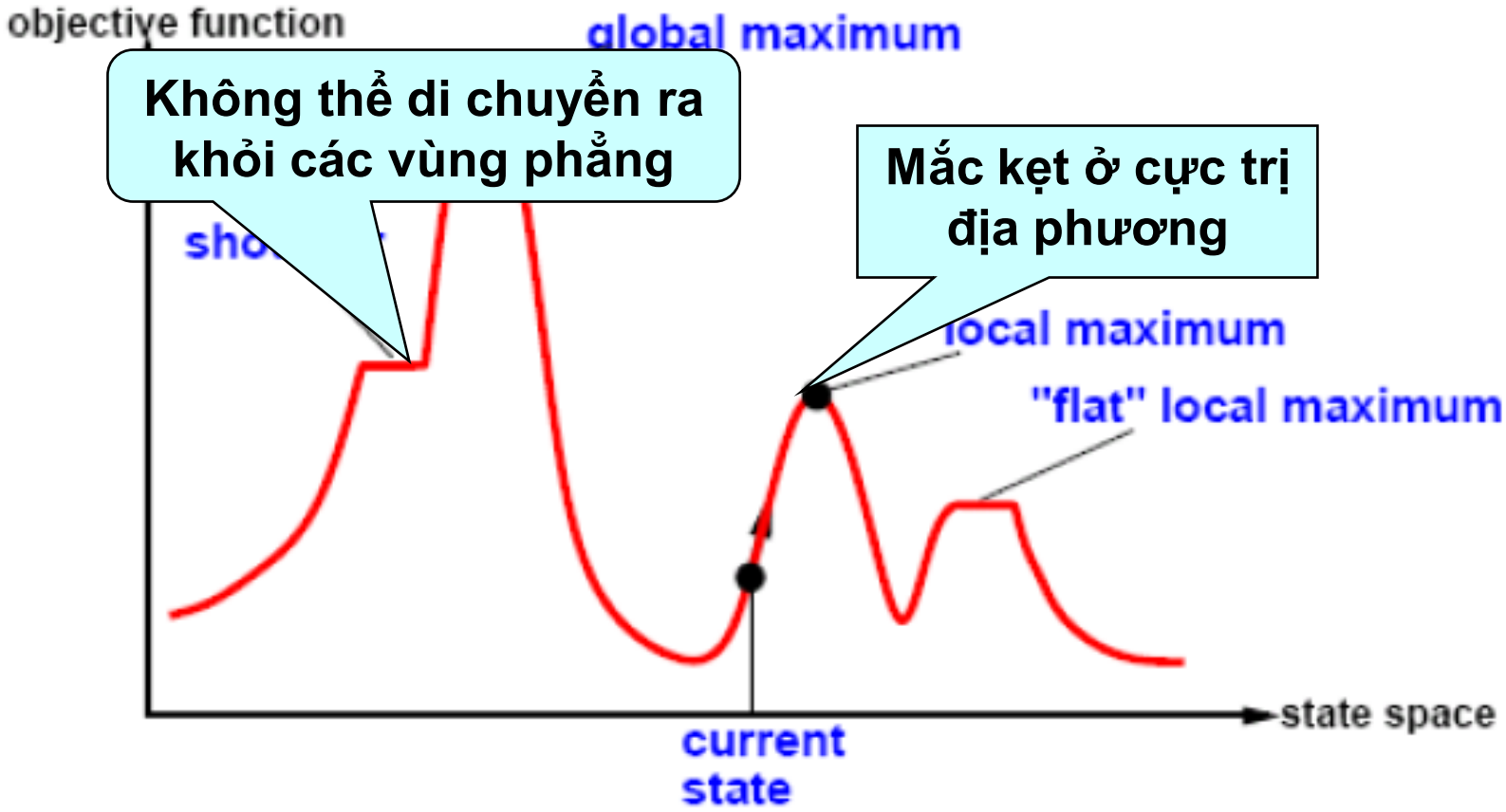
- Ví dụ: 3-change



VẤN ĐỀ CỦA LEO ĐỒI



VẤN ĐỀ CỦA LEO ĐỒI



CẢI TIẾN LEO ĐỒI

- Leo đồi với khởi tạo ngẫu nhiên nhiều lần
- Local beam search:
  - Theo dõi *k* trạng thái cùng một lúc
  - Khởi tạo với *k* trạng thái phát sinh ngẫu nhiên
  - Tại mỗi lần lặp, tất cả trạng thái con của *k* trạng thái được phát sinh
  - Nếu xuất hiện trạng thái đích thì dừng lại; ngược lại chọn *k* trạng thái con tốt nhất từ toàn bộ danh sách và lặp lại

LUYỆN THÉP

1. Đặt X := cấu hình ban đầu
2. Đặt E := Eval(X)
3. Đặt i = di chuyển ngẫu nhiên từ moveset
4. Đặt E_i := Eval(move(X,i))
5. Nếu E ≥ E_i thì
   X := move(X,i)
   E := E_i
   Ngược lại với xác suất nào đó, chấp nhận di chuyển ngay cả khi mọi chuyện xấu hơn:
   X := move(X,i)
   E := E_i
6. Quay lại 3 đến khi kết thúc.

Chúng ta sẽ chọn xác suất chấp nhận một di chuyển tồi hơn như thế nào?

- Xác suất = 0.1
- Xác suất giảm dần theo thời gian
- Xác suất
  exp (-(E - E*i*)/**T*i***): T_i là tham số nhiệt độ

Tương tự như quá trình làm lạnh trong luyện thép vật lý

THUẬT GIẢI DI TRUYỀN

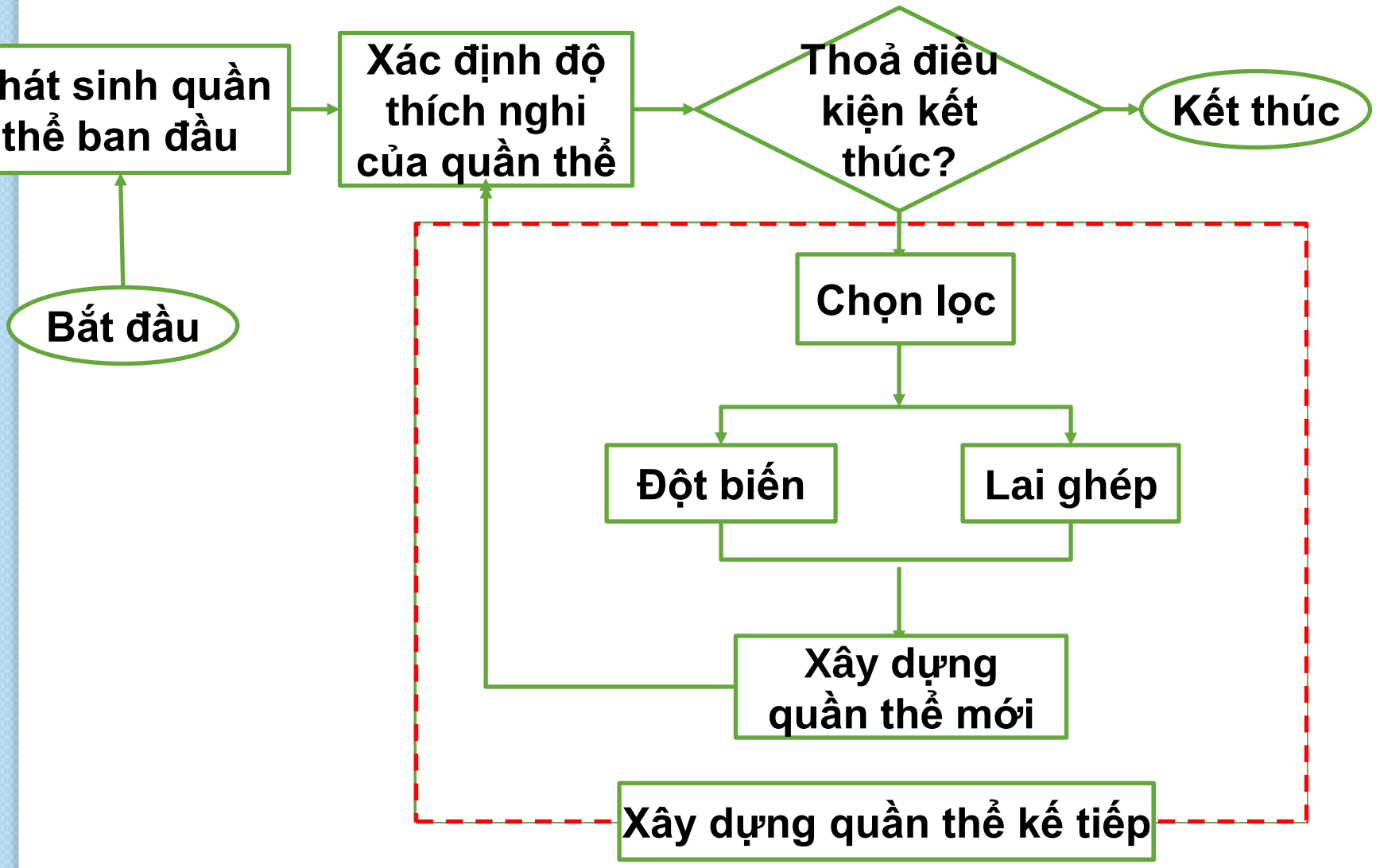
(Genetic Algorithm)

Người soạn: ThS. Nguyễn Ngọc Thảo

THUẬT GIẢI DI TRUYỀN

- Được giới thiệu bởi John Holland năm 1975, cho phép thực hiện tìm kiếm ngẫu nhiên.
- Mã hoá các lời giải tiềm năng của bài toán bằng các nhiễm sắc thể
- Đánh giá độ tốt của các lời giải qua độ thích nghi của các nhiễm sắc thể
- Lưu trữ một quần thể các lời giải tiềm năng
- Thực hiện các phép toán di truyền để phát sinh các cá thể mới đồng thời áp dụng chọn lọc tự nhiên trên các lời giải

THUẬT GIẢI DI TRUYỀN



BIỂU DIỄN GEN

- Để có thể giải bài toán bằng thuật giải di truyền ta phải gen hóa cấu trúc dữ liệu của bài toán. Có hai cách biểu diễn gen:
  - Biểu diễn gen bằng chuỗi số nguyên (hay thực).
    - Ví dụ: Bài toán 8 hậu -> 12 53 48 67 24 22 43 77
  - Biểu diễn gen bằng chuỗi nhị phân
    - Ví dụ: Bài toán 8 hậu – biểu diễn bằng $8 \times \log_2 8$ bit
    - Làm sao biểu diễn nghiệm thực bằng chuỗi nhị phân? $\Rightarrow$ Trả lời: Rời rạc hoá miền trị với một độ chính xác cho trước

CÁC KHÁI NIỆM CƠ BẢN

- Độ tốt của một cá thể là giá trị của cá thể cho một bài toán cụ thể.
  - *Ví dụ: Trong bài toán tối ưu cực đại hàm f, nếu chọn một cá thể là một nghiệm của bài toán thì một cá thể càng tốt khi làm cho giá trị hàm càng lớn.*
- Để xác định được độ tốt của các cá thể ta cần có Hàm mục tiêu.
  - Tham số đầu vào là gen của một cá thể và trả ra một số thực.
  - Tùy theo giá trị của số thực này mà ta biết được độ tốt của cá thể đó.

CÁC KHÁI NIỆM CƠ BẢN

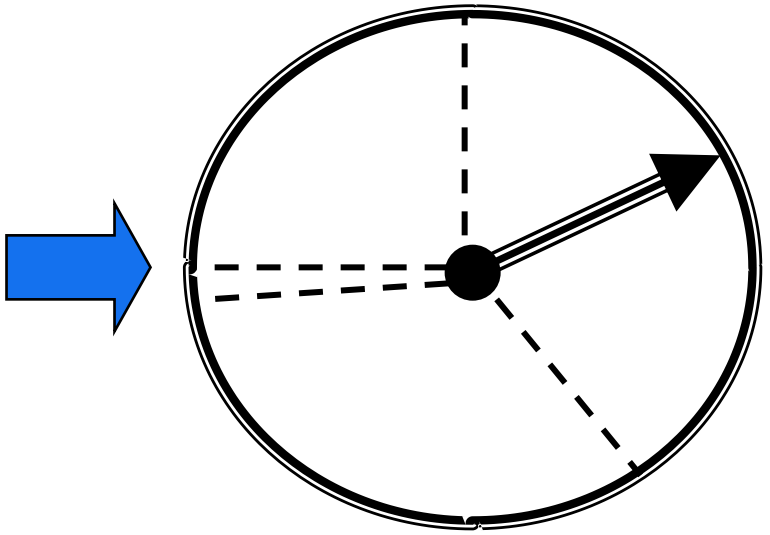
- Độ thích nghi của các cá thể (fitness)
  - Là khả năng cá thể đó được chọn lọc vào thế hệ sau hoặc là được chọn lọc cho việc lai ghép để tạo ra cá thể con .
  - Vì độ thích nghi là một xác suất để cá thể được chọn nên người ta thường ánh xạ độ thích nghi vào đoạn [0,1] (độ thích nghi chuẩn)

$$F(a_i) = \frac{F(a_i)}{\sum_{j=1}^{N} F(a_j)} \quad i = 1,2,\ldots,N$$

CÁC TOÁN TỬ CƠ BẢN

- Toán tử lai ghép
  - Các cá thể được chọn để lai ghép dựa vào dựa vào độ thích nghi
  - Dùng qui tắc bàn quay roullette

STT	Cá thể	ĐTN chuẩn
1	0010001	0.4
2	0010101	0.3
3	0101000	0.05
4	1100011	0.25

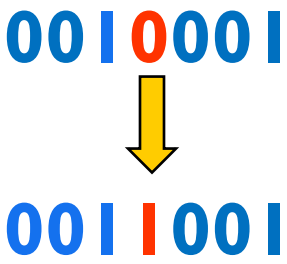


CÁC TOÁN TỬ CƠ BẢN

- Toán tử lai ghép
  - Lấy giá trị ngẫu nhiên $p \in [0,1]$ để chọn cá thể lai ghép, cá thể có độ thích nghi cao có xác xuất lựa chọn nhiều hơn
  - Sau khi lựa chọn một cặp cá thể cha mẹ, hoán vị các nhiễm sắc thể tại vị trí ngẫu nhiên với xác suất p_c
- Toán tử lai ghép có xu hướng kéo quần thể về phía các cá thể có độ thích nghi cao $\Rightarrow$ cục bộ địa phương

CÁC TOÁN TỬ CƠ BẢN

- Toán tử đột biến
  - Giúp lời giải có thể nhảy ra khỏi các cực trị địa phương
  - Với mỗi cá thể trong quần thể, thực hiện đột biến với xác suất p_m tại một vị trí ngẫu nhiên (thông thường $p_m << 0.1$)



CÁC TOÁN TỬ CƠ BẢN

- Xác định kích thước quần thể: **n = 4**
- Chọn phương pháp mã hóa nghiệm:
  - Xác định nghiệm nguyên trong miền trị: [0, 31]
  - Mã hoá theo chuỗi nhị phân: số bit mã hoá = 5
- Lựa chọn hàm thích nghi
  - Hàm thích nghi = 1000 – (X^2 – 64), chọn nghiệm có hệ số thích nghi ~ 1000

GIẢI PHƯƠNG TRÌNH BẬC 2

$$X^2 = 64$$

- Xác định kích thước quần thể: **n = 4**
- Chọn phương pháp mã hóa nghiệm:
  - Xác định nghiệm nguyên trong miền trị: [0, 31]
  - Mã hoá theo chuỗi nhị phân: số bit mã hoá = 5
- Lựa chọn hàm thích nghi
  - Hàm thích nghi = 1000 – (X^2 – 64), chọn nghiệm có hệ số thích nghi ~ 1000

GIẢI PHƯƠNG TRÌNH BẬC 2

$$X^2 \equiv 64$$

- Xác định kích thước quần thể: 4
- Chọn phương pháp mã h
  - Xác định nghiệ 31]
  - Mã hoá theo chu ã hoá = 5
- Lựa chọn hàm th
  - Hàm thích ng – 64), chọn nghiệm có hệ số thích ng 0

Các bước sau đây được thực hiện dựa vào "ngẫu nhiên"

GIẢI PHƯƠNG TRÌNH BẬC 2

$$X^2 = 64$$

- Phát sinh tập quần thể ban đầu

STT	Nhị phân	Nghiệm
1	00100	4
2	10101	21
3	01010	10
4	11000	24

GIẢI PHƯƠNG TRÌNH BẬC 2

$$X^2 = 64$$

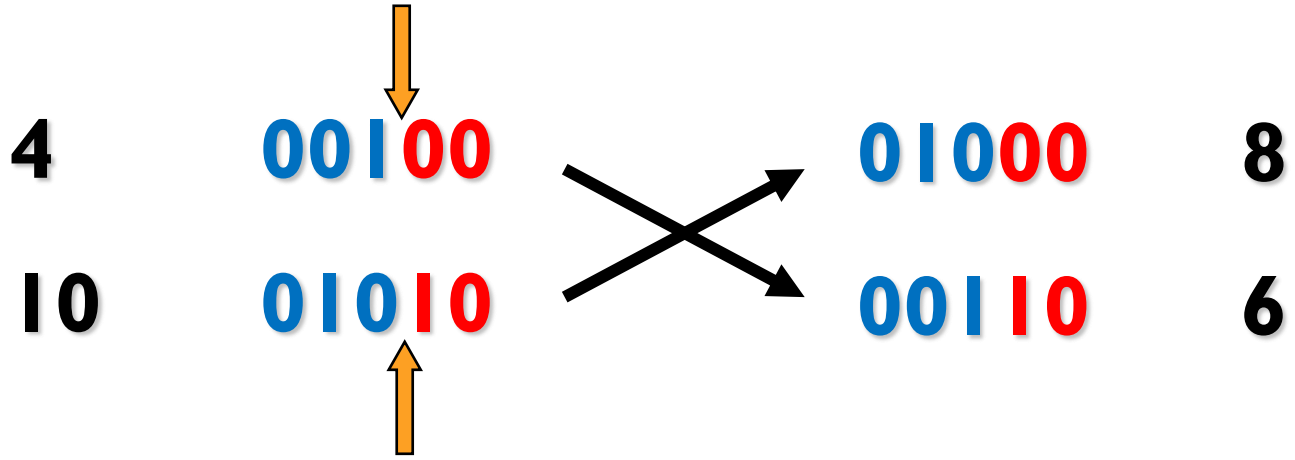
- Tính hệ số thích nghi (Fitness) cho quần thể

STT	Nhị phân	Nghiệm	$X^2 - 64$	Hệ số thích nghi
1	00100	4	-48	1048
2	10101	21	377	623
3	01010	10	36	964
4	11000	24	512	488

GIẢI PHƯƠNG TRÌNH BẬC 2

$$X^2 = 64$$

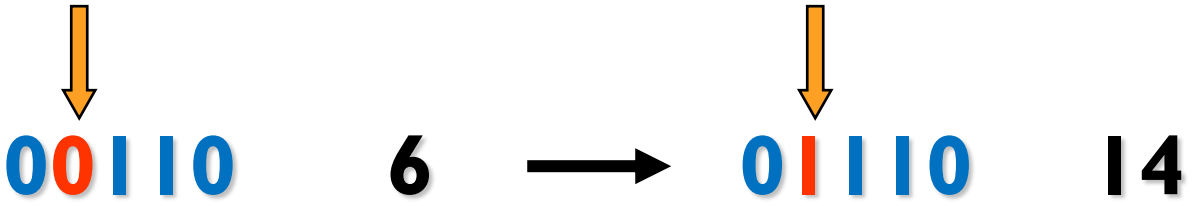
- Chọn lọc nghiệm và lai ghép
  - Ví dụ: *Chọn nghiệm 4 và 10 để tiến hành lai ghép với xác suất p_c và vị trí pos= 2*



GIẢI PHƯƠNG TRÌNH BẬC 2

$$X^2 = 64$$

- Đột biến cá thể
  - Ví dụ: *Với một xác suất p_m đột biến lời giải thứ 4 với vị trí pos = 4*



GIẢI PHƯƠNG TRÌNH BẬC 2

$$X^2 = 64$$

- Tính lại hệ số thích nghi cho nghiệm mới và tiến hành chọn lọc

STT	Nhị phân	Nghiệm	$X^2 - 64$	Hệ số thích nghi
1	00100	4	-48	1048
2	01010	10	36	964
3	01000	8	0	1000
4	01110	14	132	868

TỔNG KẾT

- Nắm vững thuật toán Greedy Best First Search và nhược điểm
- Nắm vững thuật toán A*, các trường hợp cập nhật trạng thái, và điều kiện dừng của A*.
- Biết "tính chấp nhận được" của heuristic. Chứng minh tính đầy đủ, đảm bảo tối ưu của đường đi.
- Biết tư tưởng của thuật giải leo đồi và các cải tiến
- Biết các bước thực hiện của thuật giải di truyền, khái niệm các toán tử cơ bản.

TÀI LIỆU THAM KHẢO

- Tài liệu bài giảng môn học
- Moore's tutorial:

http://www.cs.cmu.edu/~awm/tutorials

- Chapter 3. S. Russel and P.Norvig, Artificial Intelligence – A Modern Approach. *Third Edition.* 2010

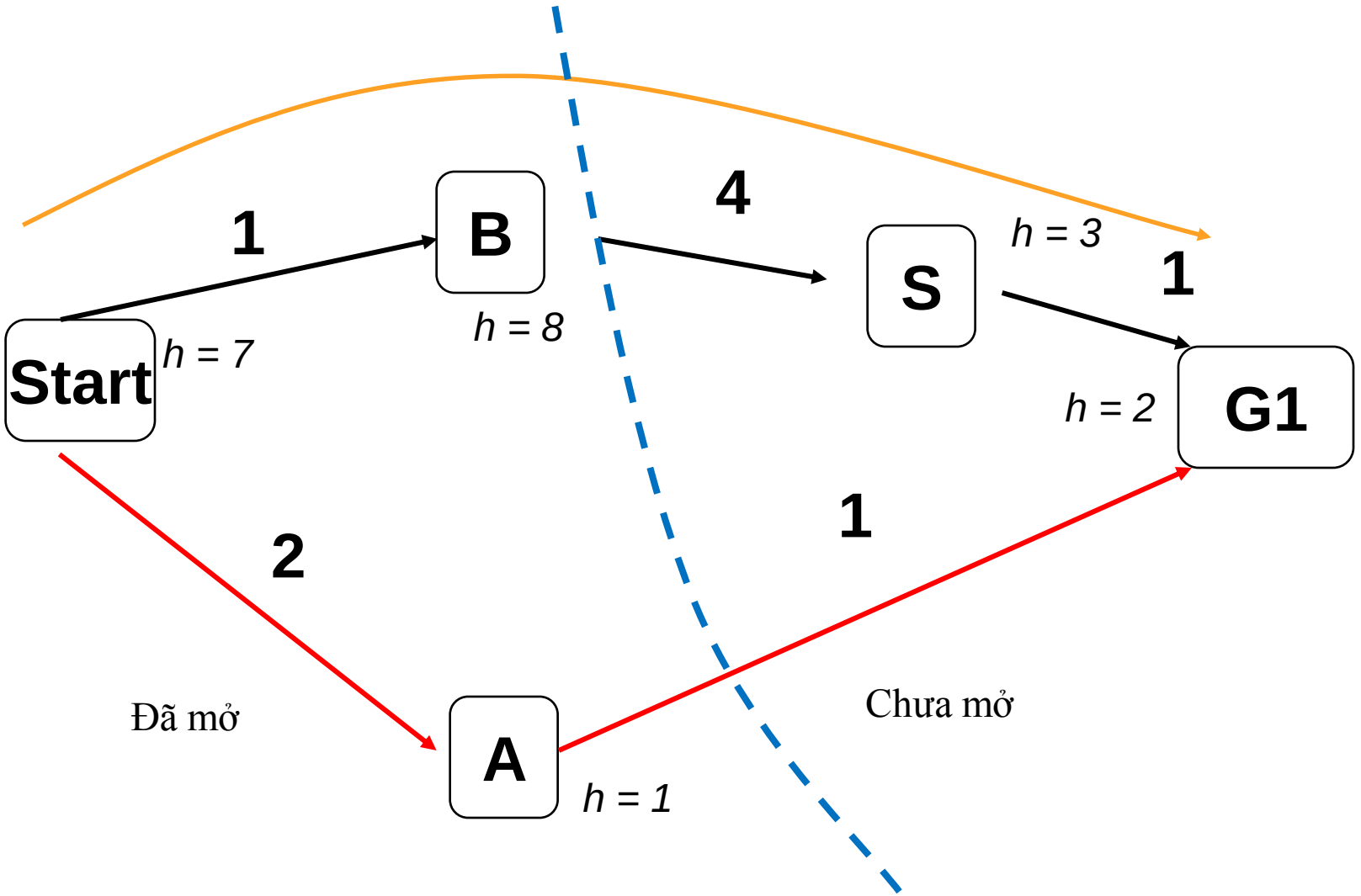
Chứng minh: A* Heuristic Chấp nhận được Bảo đảm Tối ưu

- Giả sử nó tìm thấy đường đi không tối ưu, kết thúc tại trạng thái đích G_1 trong đó $f(G_1) > f^*$ với $f^* = h^*(start)$ = chi phí đường đi tối ưu.
- Phải tồn tại một node s
  - Chưa mở
  - Đường đi từ điểm đầu đến s (lưu trong các giá trị BackPointers(s)) là bắt đầu của đường đi tối ưu thật sự
- $f(s) \geq f(G_1)$ (ngược lại tìm kiếm đã không kết thúc)
- Cũng thế

$$f(s) = g(s) + h(s) = g^*(s) + h(s) \leq g^*(s) + h^*(s) = f^*$$

- Do đó $f(s) = f^* \geq f(G_1)$ (vô lý)
- *Vậy: đường đi tìm thấy là đường đi tối ưu* $f^* = f(G_1)$

Chứng minh: A* Heuristic Chấp nhận được Bảo đảm Tối ưu



Chứng minh: A* Heuristic Chấp nhận được Bảo đảm Tối ưu

- Giả sử nó tìm thấy đường đi không tối ưu, kết thúc tại trạng thái đích G_1 trong đó $f(G_1) > f^*$ với $f^* = h^*(start)$ = chi phí đường đi tối ưu.
- Phải tồn tại một node s
  - Chưa mở
  - Đường đi từ điểm đầu đến s
    BackPointers(s)) là bắt đầu của đu
- $f(s) \geq f$ m kiếm đã kh
- Cũng thế

$$f(s) = g(s) + h(s) = g^*(s) + h(s) \leq g^*(s) + h^*(s) = f^*$$

- Do đó $f(s) = f^* \geq f(G_1)$ (vô lý)
- *Vậy: đường đi tìm*

Tại sao một node như thế phải tồn tại? Xem xét bất kỳ đường đi tối ưu *s*,*s1*,*s2*…goal. Nếu các node dọc theo nó được mở, đích sẽ được đi đến theo đường đi ngắn nhất.

vì nó nằm trên đường đi tối ưu

mâu thuẫn

Do giả thiết chấp nhận được

Vì s nằm trên đường đi tối ưu

Demo

- [1] Genetic Algorithms Explanation
- [2] RobotCode
- [3] 3D Travelling Salesman Problem (TSP)
- [4] Mario – Genetic Algorithms

  Hàm thích nghi: *(Distance/MaxDistance)*100 - (time/10)*

  (càng lớn càng tốt)
- [5] Airman vs Genetic Algorithms

  Hàm thích nghi:Megaman health – Airman health

  (càng lớn càng tốt)
- [6] GA Crawler (learning how to walk)

KẾT THÚC CHỦ ĐỀ