TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA TOÁN - CƠ - TIN HỌC

KHÓA LUẬN TỐT NGHIỆP

Phương pháp rời rạc động giải các bài toán tìm đường đi có yếu tố thời gian

Sinh viên: Bùi Khánh Duy

Giảng viên hướng dẫn: TS. Vũ Đức Minh

Hà Nội - 07/06/2024

Bố cục

1. Giới thiệu bài toán
2. Mô tả bài toán
3. Chi tiết thuật toán
4. Minh họa thuật toán
5. Kết quả thử nghiệm, kết luận

Giới thiệu

Tìm đường đi nhỏ nhất?

- Bài toán tìm đường đi nhỏ nhất: tìm một đường đi giữa hai đỉnh sao cho tổng các trọng số của các cạnh tạo nên đường đi đó là nhỏ nhất.

- Tìm đường đi nhỏ nhất có thể là thời gian đến / thời gian xuất phát / thời gian di chuyển nhỏ nhất.

- Cần giải quyết bài toán

Tìm đường đi có thời gian di chuyển tối thiểu trong mạng phụ thuộc thời gian - Minimum Duration Time-Dependent Path Problem (MDP)

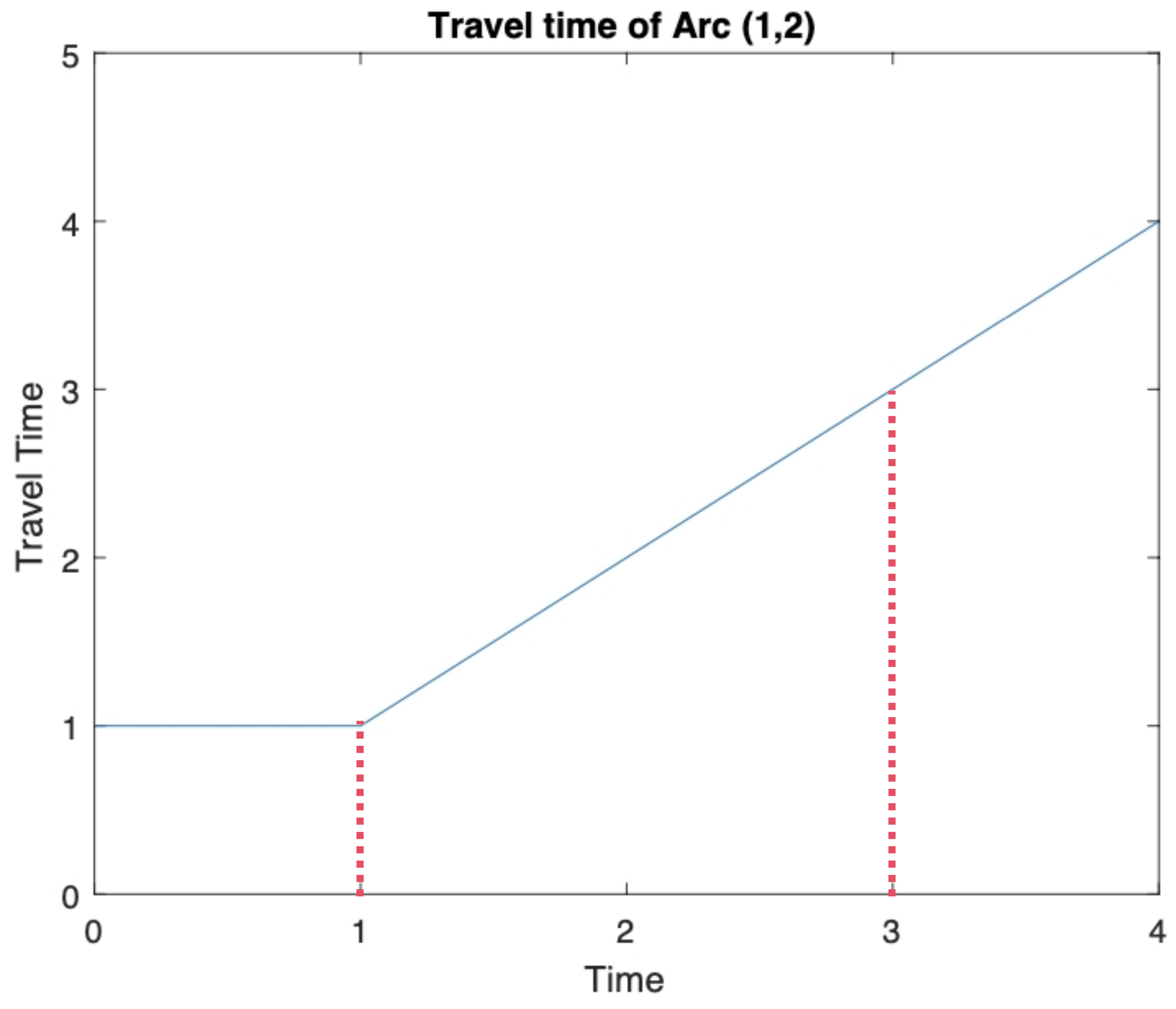
Giới thiệu

Mạng phụ thuộc thời gian - Time-dependent network

- Bài toán tìm đường đi ngắn nhất trong mạng phụ thuộc thời gian gọi là TDSPP (time-dependent shortest path problem)

- Do tắc nghẽn khác nhau trên các con đường trong ngày, cả thời gian di chuyển từ điểm nguồn s đến đích d và đường đi tối ưu giữa chúng có thể thay đổi theo thời gian.

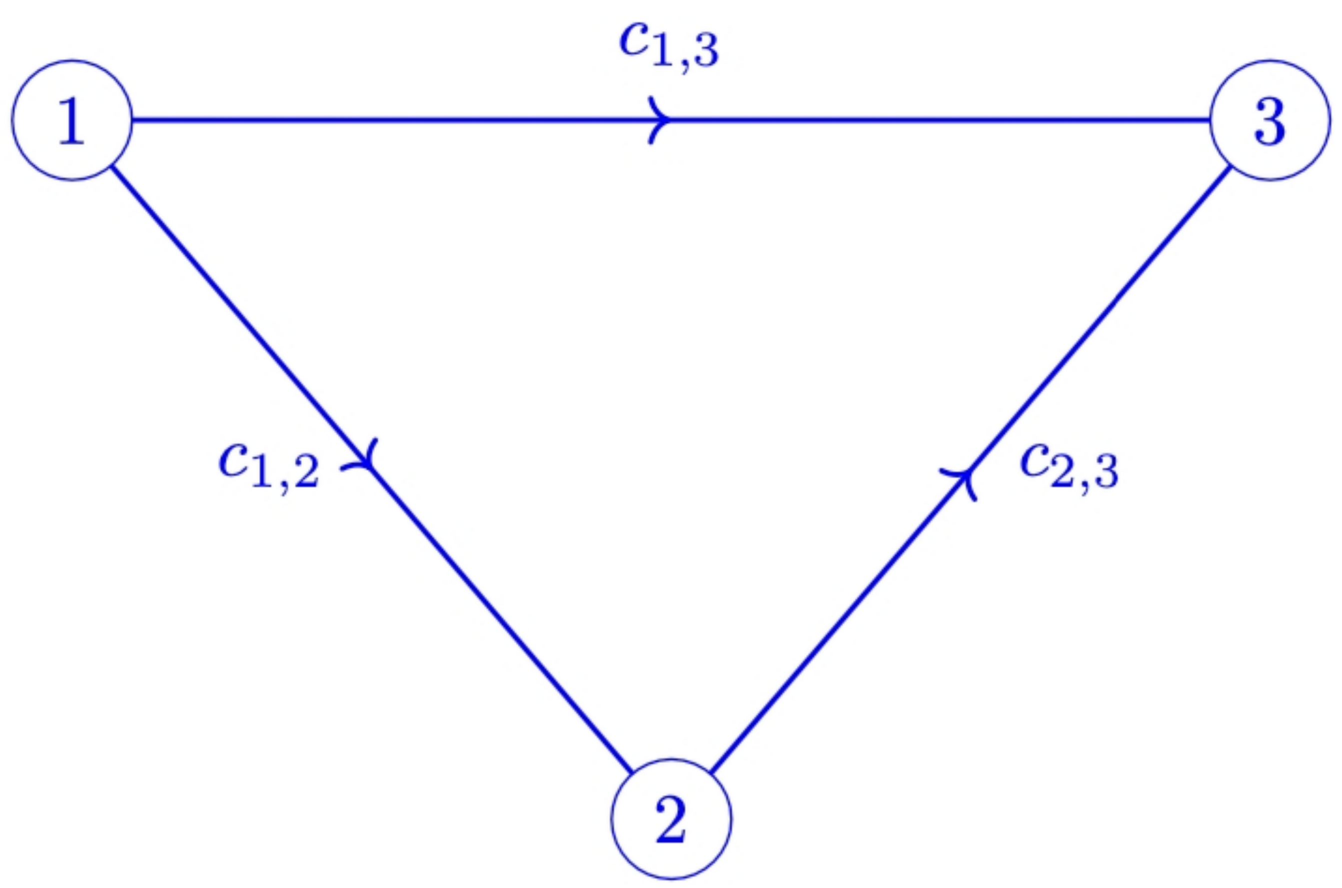
- Trong thực trạng giao thông, có những tuyến đường bị tắc vào giờ cao điểm. Vậy nên việc đổi thời gian khởi hành có thể giảm thời gian phải đi.



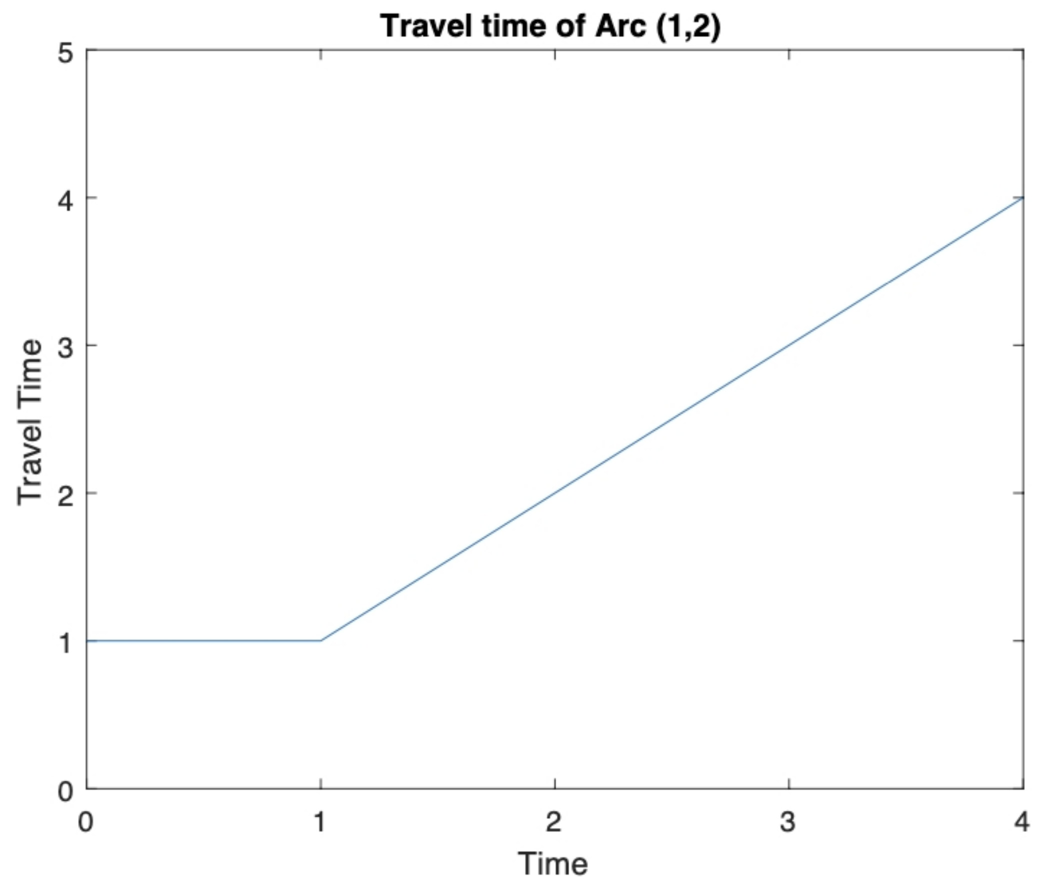
Giới thiệu

Hàm thời gian di chuyển tuyến tính từng khúc - Piecewise linear arc travel time functions

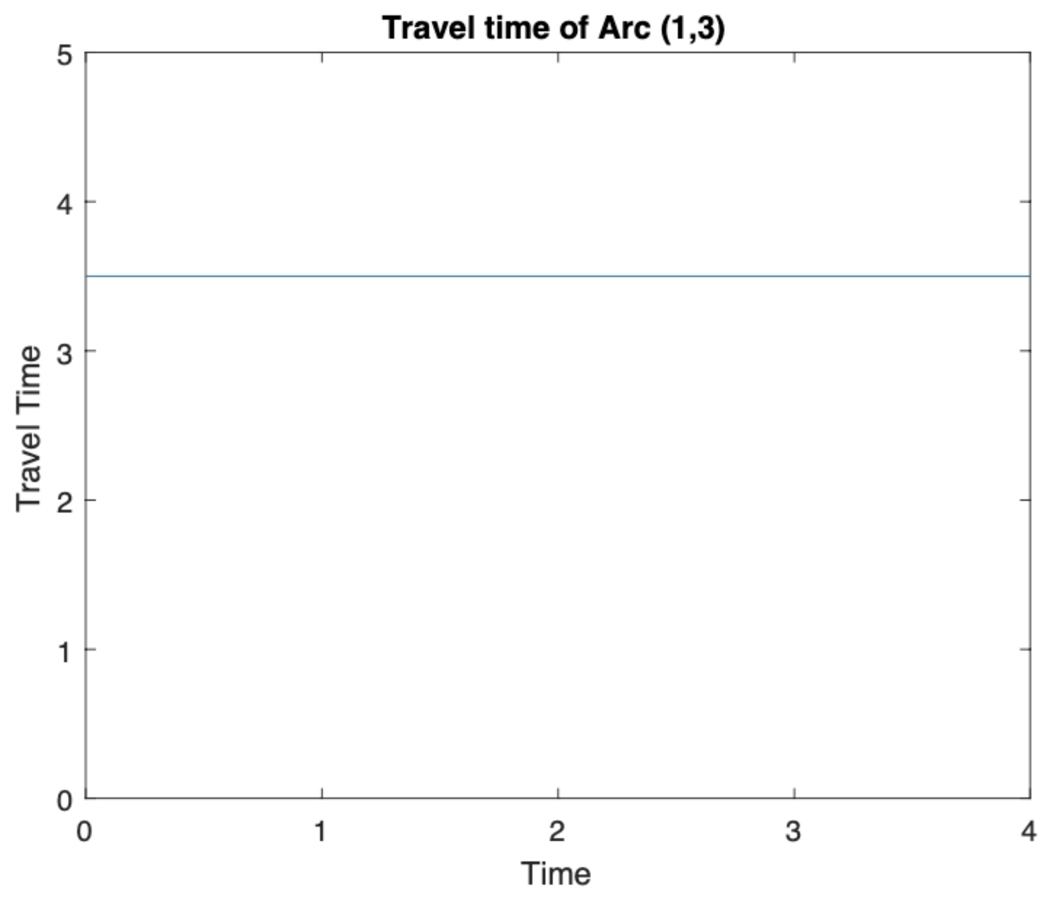
- Mô tả sự thay đổi trong mạng phụ thuộc thời gian bằng $c_{i,j} > 0$ ((i, j) là cung thuộc mạng)
- Trọng số (mạng tĩnh) => hàm thời gian di chuyển (mạng phụ thuộc thời gian)
- Ví dụ 1:



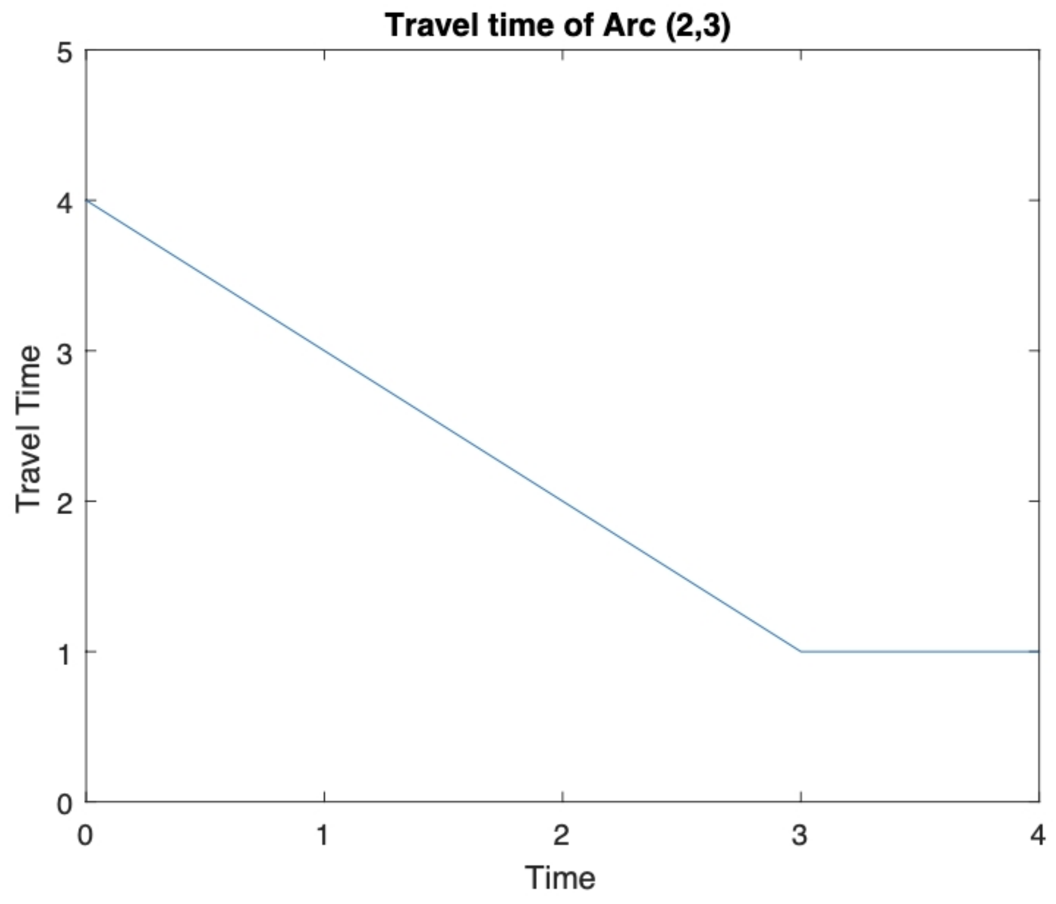
Đồ thị VD 1



Cung (1,2)



Cung (1,3)



Cung (2,3)

Tính chất: FIFO

Vào trước ra trước - First in First out

- Mạng phụ thuộc thời gian thường được giả định là có tính chất FIFO.
- Việc di chuyển trên các cung tuân theo quy tắc

  không thể đến cuối cung sớm hơn nếu vào cung muộn hơn

- Một số **tính chất**:
  - Việc chờ đợi tại các nút không có lợi (vì không làm giảm thời gian đến đích).
  - Luôn có thể tìm thấy các đường đi ngắn nhất không là chu trình.

Các bài báo liên quan

• 2011 Demiryurek et al	Nghiên cứu bài toán MDP trong bối cảnh mạng lưới giao thông.
• 2012 Gunturi et al	Sử dụng bài toán MDP để phân tích mạng xã hội.
• 2004 Dean et al	Nghiên cứu về tìm đường đi ngắn nhất của mạng phụ thuộc thời gian có tính chất FIFO
• 2017 Boland et al	Nghiên cứu phương pháp rời rạc động (DDD) để giải mạng phụ thuộc thời gian
• 2018 Foschini	Phương pháp vét cạn (Enum) các BP để giải bài toán MDP
• 2018 E. Y. He et al • 2022 E. Y. He et al	Sử dụng DDD để giải bài toán MDP và một số bài toán tương tự

Mô tả bài toán

- Đầu vào
- Đầu ra
- Hàm mục tiêu
- Các thao tác cần giải quyết
- Mã giả

Mô tả bài toán - MDP

Đầu vào

Đồ thị có hướng

$$G = (N, A)$$

trong đó:

- $N = 1,2,...,n$ là tập đỉnh
- $A \subseteq N \times N$ là tập cạnh
- $[0,T]$ là khung thời gian
- Các hàm thời gian di chuyển $c_{i,j}(t) > 0$ với $t \in [0,T]$ thỏa mãn tính chất FIFO cho mọi cạnh $(i, j) \in A$

Mô tả bài toán - MDP

Đầu ra

Một đường đi xuất phát từ đỉnh nguồn 1, kết thúc ở đỉnh đích n có dạng như sau

$$P = ((i_1, t_1), (i_2, t_2), \ldots, (i_n, t_n))$$

Các nút cần thỏa mãn tính chất:

$$t_{k+1} \geq t_k + c_{i_k, i_{k+1}}(t_k)$$

với $k = 1, \ldots n - 1$

- Nếu $t_k > t_{k-1} + c_{i_{k-1}, i_k}(t_{k-1})$ với $k \geq 2$ thì đỉnh i_k cần phải đợi
- Nếu $t_k = t_{k-1} + c_{i_{k-1}, i_k}(t_{k-1})$ với $k \geq 2$ bất kì thì P **không có chờ đợi**

Mô tả bài toán - MDP

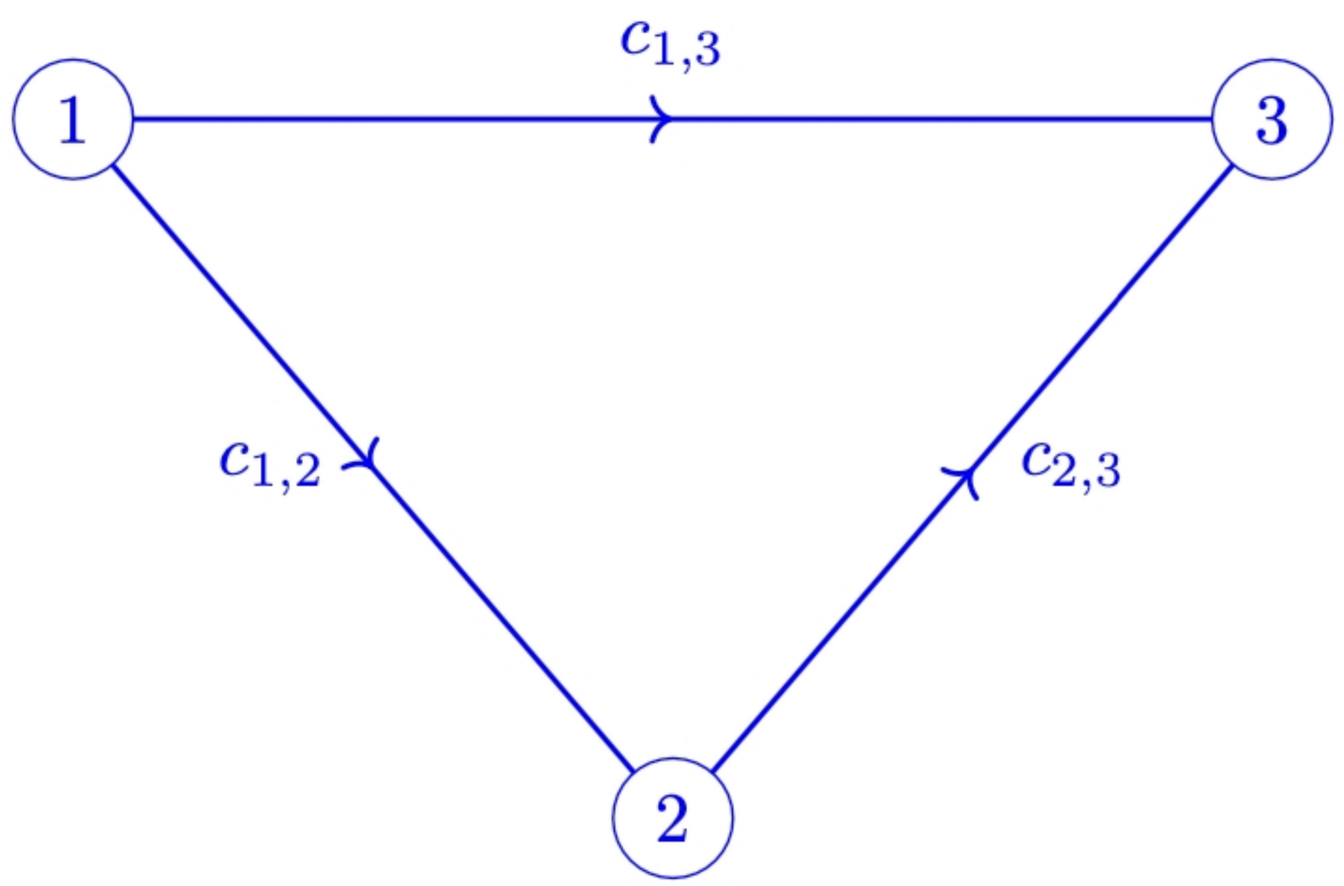
Hàm mục tiêu

Tối thiểu thời gian thực hiện (đi từ nguồn đến đích)

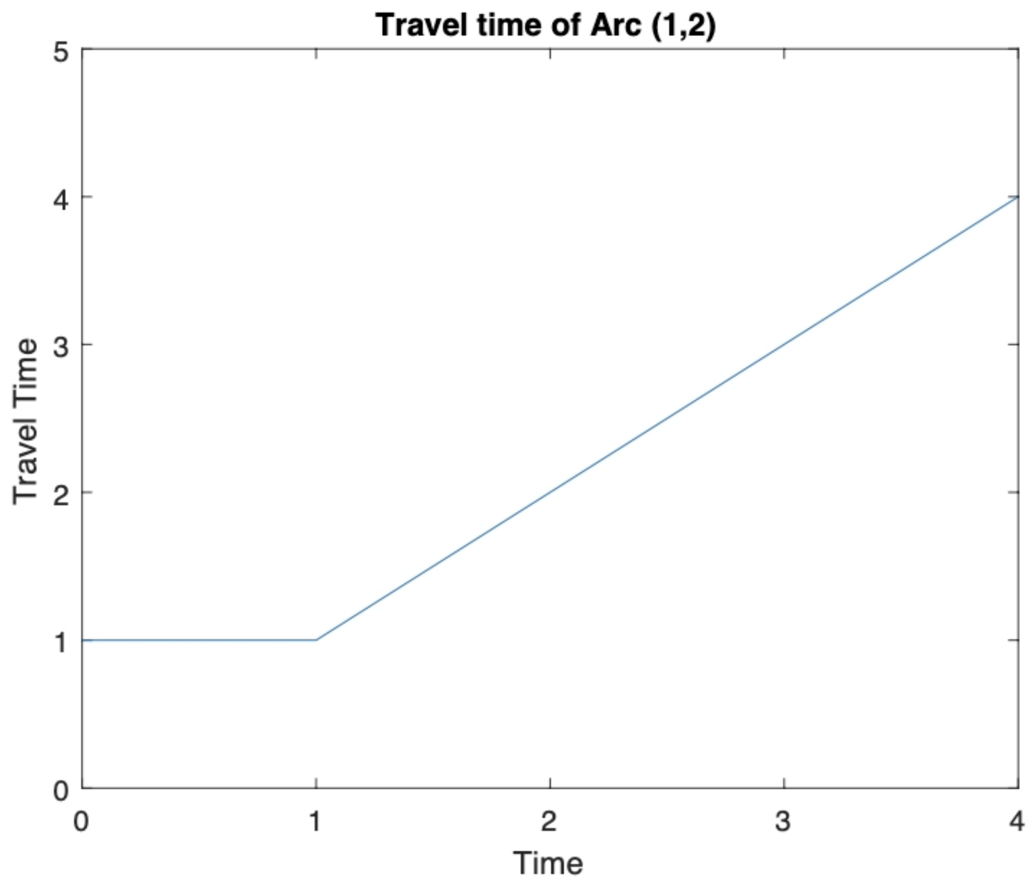
$$\min_{P \in \mathcal{P}} (t_n - t_1)$$

Ví dụ 1

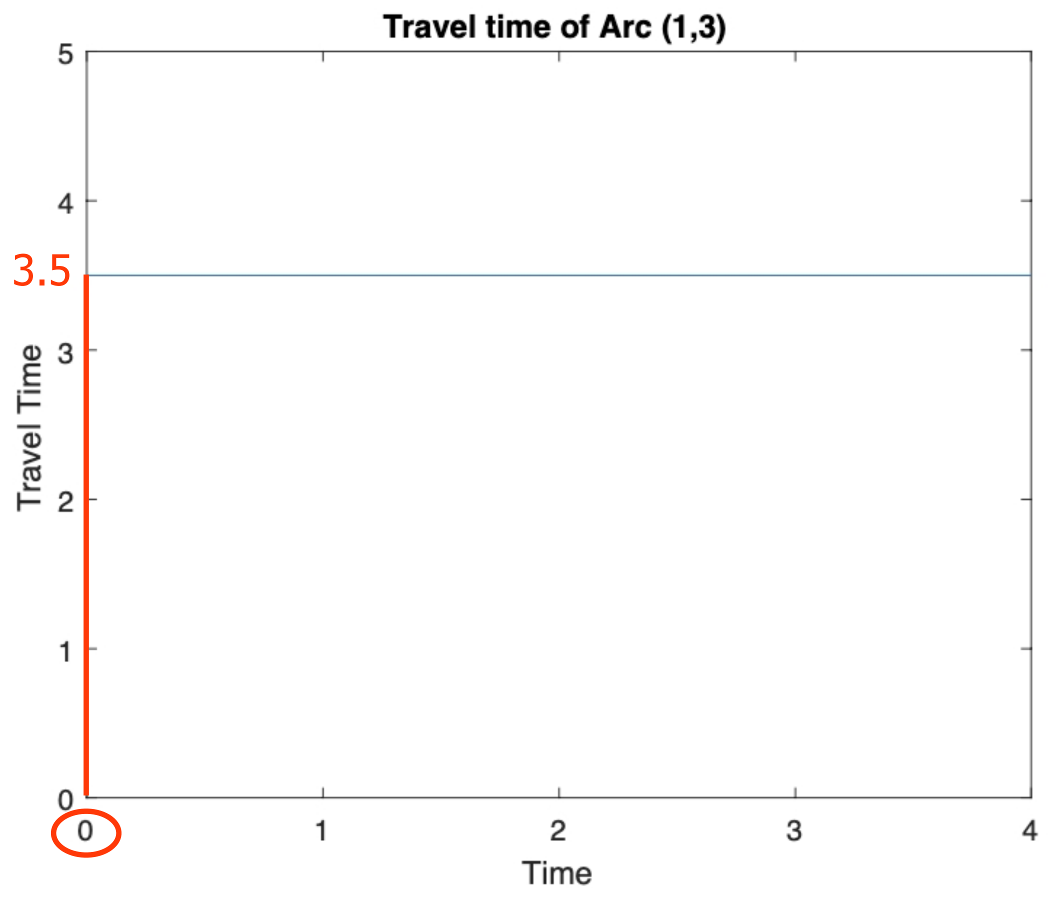
Đồ thị, các hàm thời gian di chuyển



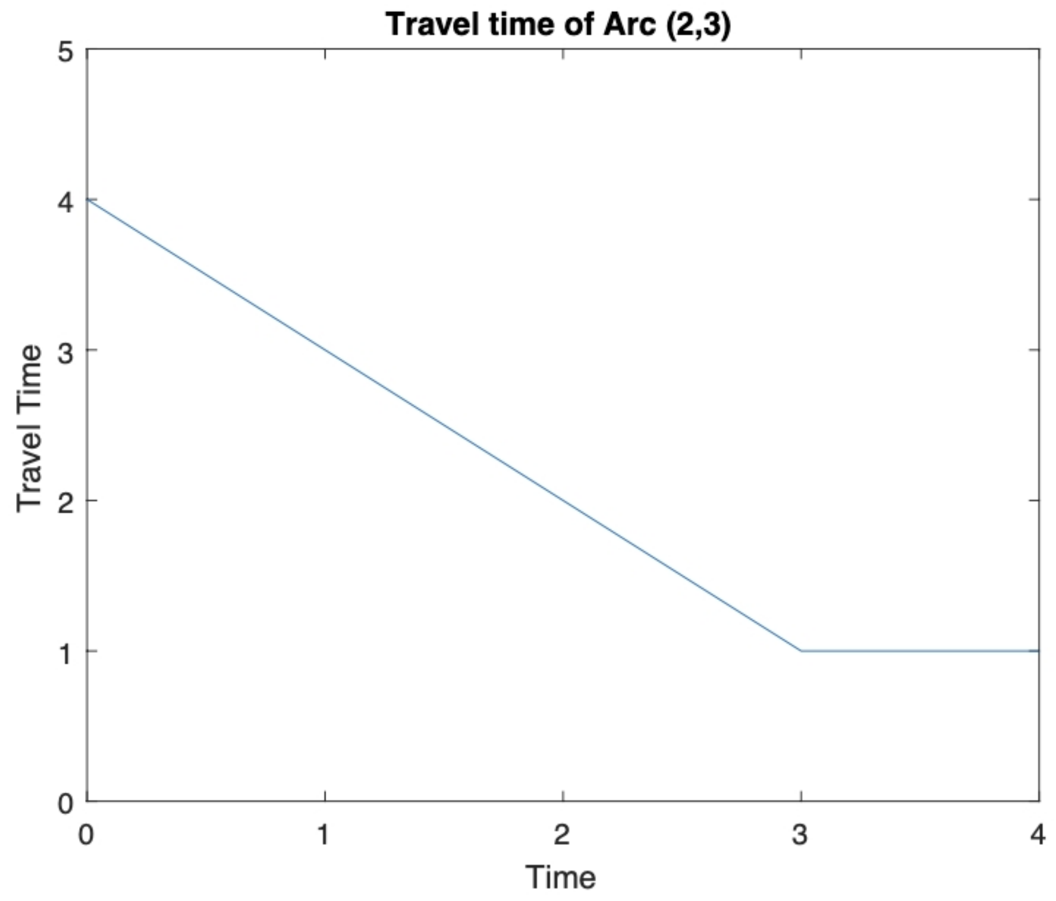
Đồ thị VD 1



Cung (1,2)



Cung (1,3)



Cung (2,3)

- Đường đi **tối thiểu thời gian thực hiện**: $[(1,1.5),(2,3),(3,4)]$, $t_n - t_1 = 2.5$

Mô tả bài toán - MDP

Các thao tác giải quyết

1. Xác định một ABSPT bất kì từ danh sách, giả sử là $\mathcal{D}^{(j,t)}$ gốc là nút (j, t).
2. Tính toán và cập nhật các LB, UB toàn cục:
    1. Hàm cận trên $computeUB(\mathcal{D}^{(j,t)}) = t_n - t_1$
    2. Hàm cận dưới $computeLB(\mathcal{D}^{(j,t)})$: Tìm thời gian di chuyển nhỏ nhất (UTT) trong $\mathcal{D}^{(j,t)}$ từ $(1, t_1)$ đến (n, t_n) và trả kết quả
    3. Lưu lại các giá trị

        $LB^{j,t} \leftarrow computeLB(\mathcal{D}^{(j,t_j)}), UB^{j,t} \leftarrow computeUB(\mathcal{D}^{(j,t_j)})$

        $LB = updateLB(\cdot), UB = \min(UB, UB^{(j,t)})$
3. Đối với ABSPT cuối cùng trong danh sách: $\mathcal{D}^{(n,T)}$:
    1. Các UTT sẽ là thời gian di chuyển thực tế nếu cung có trong BSPT, và là vô cực trong các trường hợp khác,
    2. Cận dưới $LB^{(n,T)} = UB^{(n,T)}$ (bằng cận trên).

Chi tiết

Mã giả

Algorithm 1: Dynamic Discretization Discovery (DDD) Algorithm.

input : $G = (N, A)$, $c_{i,j}(t)$, T

output: minimum duration shortest path

```
L ← (D^(1,0), D^(n,T));
UB ← min{computeUB(D^(1,0)), computeUB(D^(n,T))} ;
LB ← computeLB(D^(1,0)) ;
D^(1,t_1^k) ← D^(1,0) ;
while (LB < UB) do
    if there is a breakpoint (j,τ) between D^(1,t_1^k) and D^(1,t_1^(k+1)) then
        if computeUB(D^(j,τ)) < UB then
            UB ← UB(D^(j,τ))
        end
        recomputeLB(D^(1,t_1^k)) ;
        computeLB(D^(j,τ)) ; insert(L, D^(j,τ)) ;
    else
        LB(D^(1,t_1^k)) = UB(D^(1,t_1^k)) ;
    end
    LB ← updateLB(L) ;
    D^(1,t_1^k) ← getBestLB(L) ;
end
```

Độ phức tạp thời gian:

$\mathcal{O}(K \times SSP)$ với K là số BP

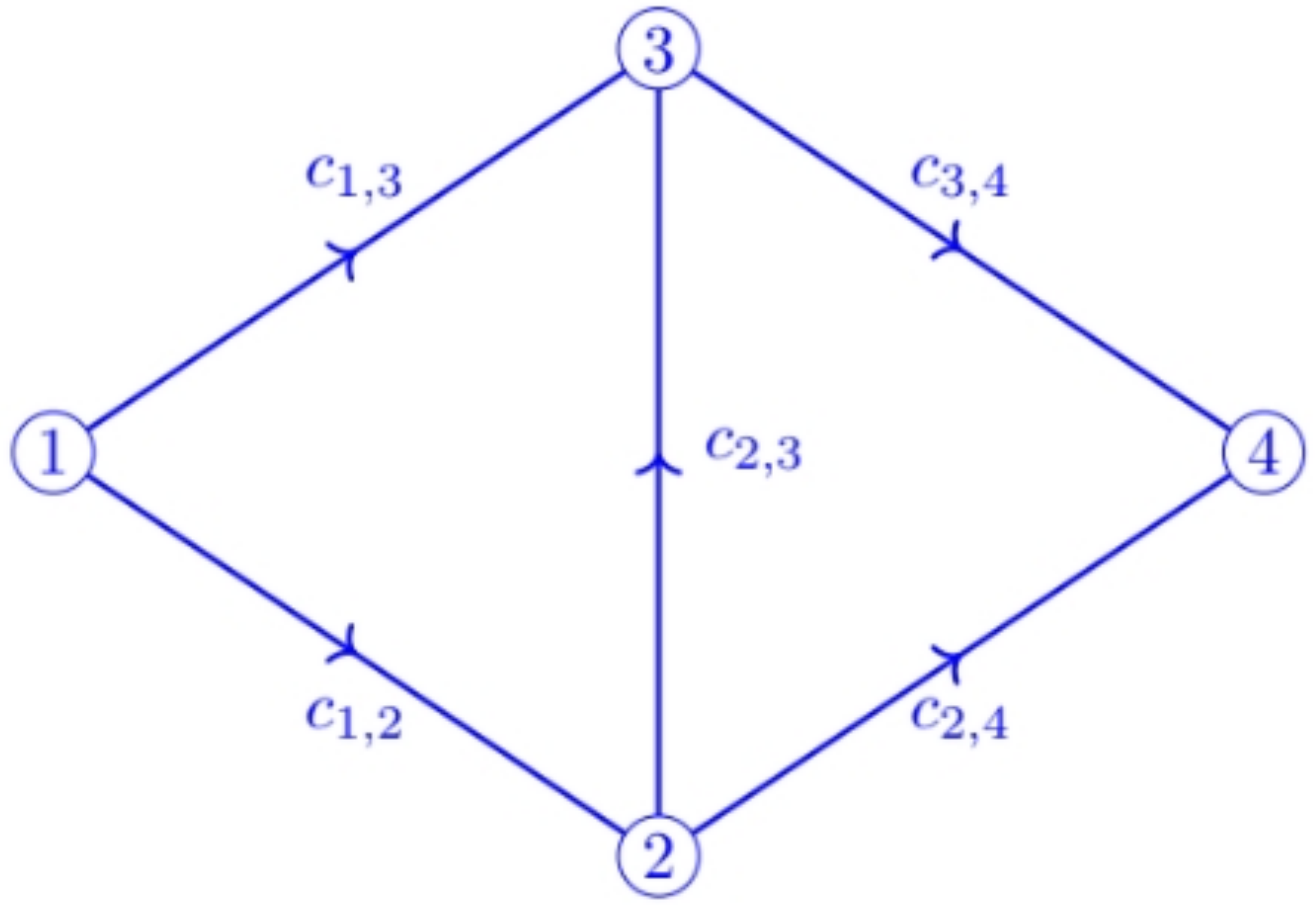
Khám phá rời rạc động

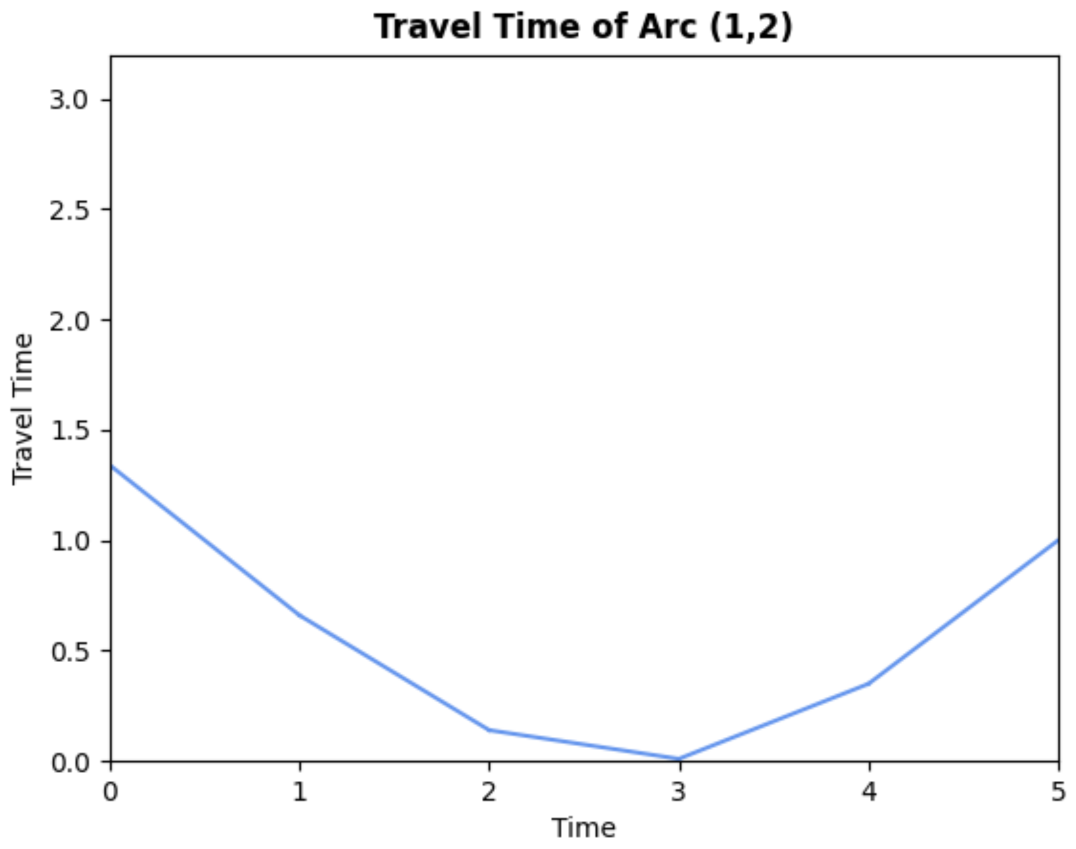
DDD - Dynamic Discretization Discovery

- Xét các thời điểm hàm di chuyển thay đổi độ dốc (gọi là BP - breakpoint)
- Các vấn đề chính của phương pháp:
    - Tìm các **thời điểm** thích hợp nhất
    - Tạo mạng thời gian từng phần (**TEN** - Time-expanded network) (ở đây là ABSPT)
    - Tìm ra các cận trên - dưới
    - Lặp lại các bước để hai cận hội tụ, tìm được giá trị tối ưu

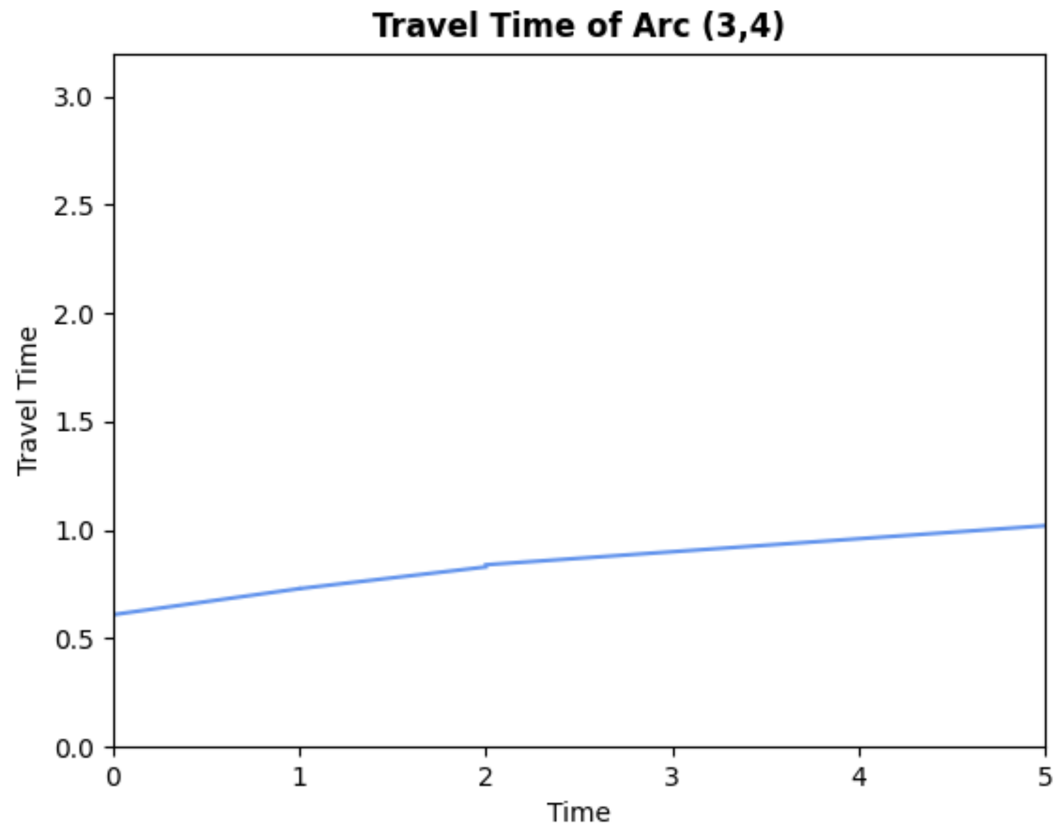
Ví dụ 2

Đồ thị, các hàm thời gian di chuyển



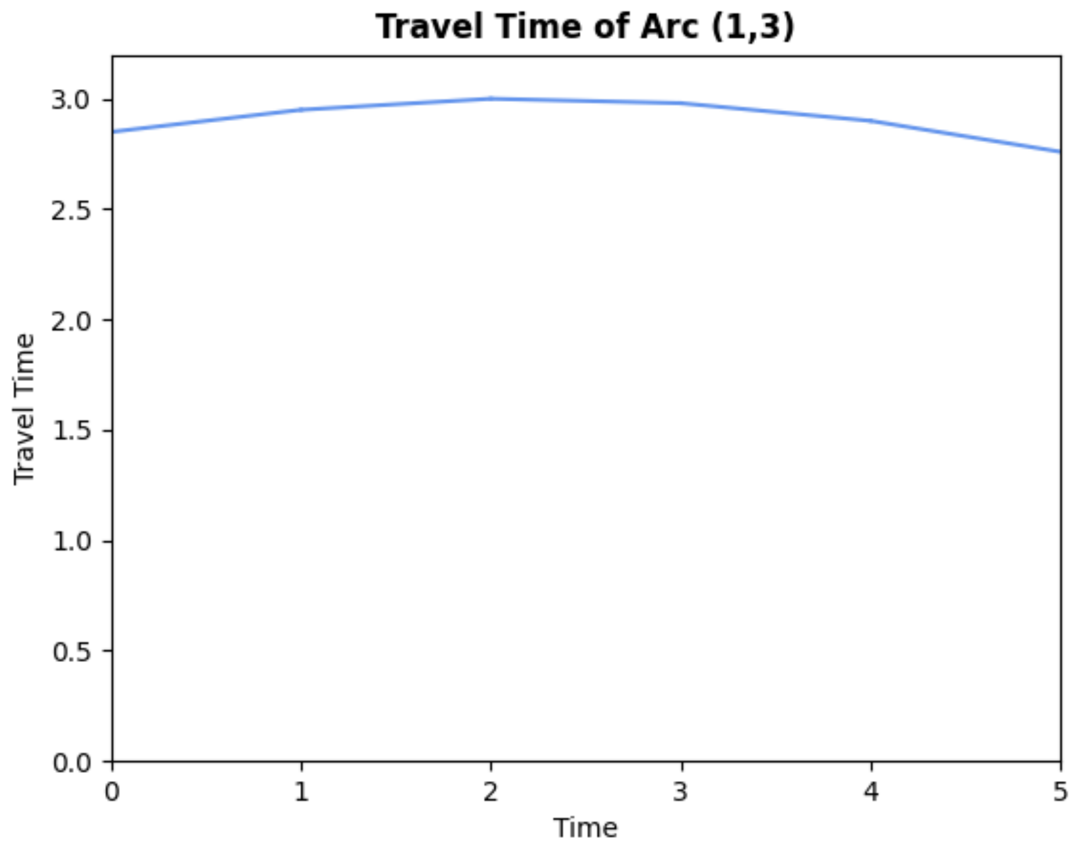


Cung (1,2)

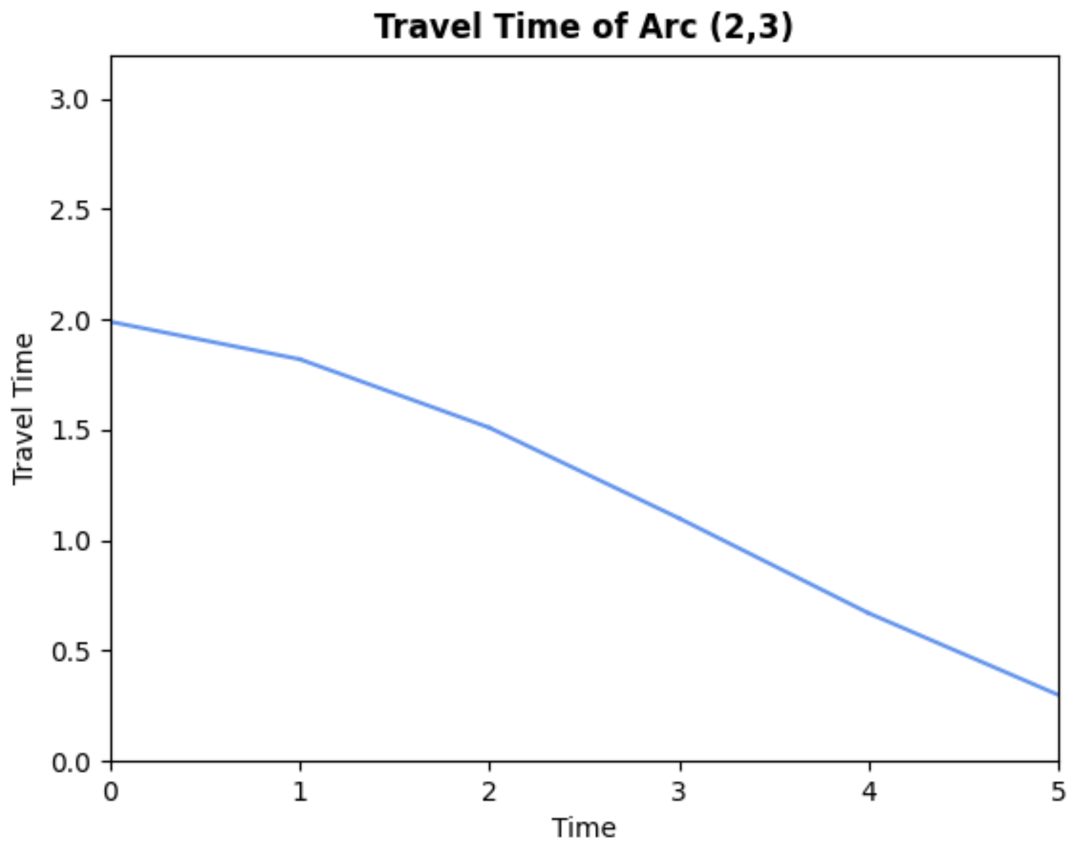


Cung (3,4)

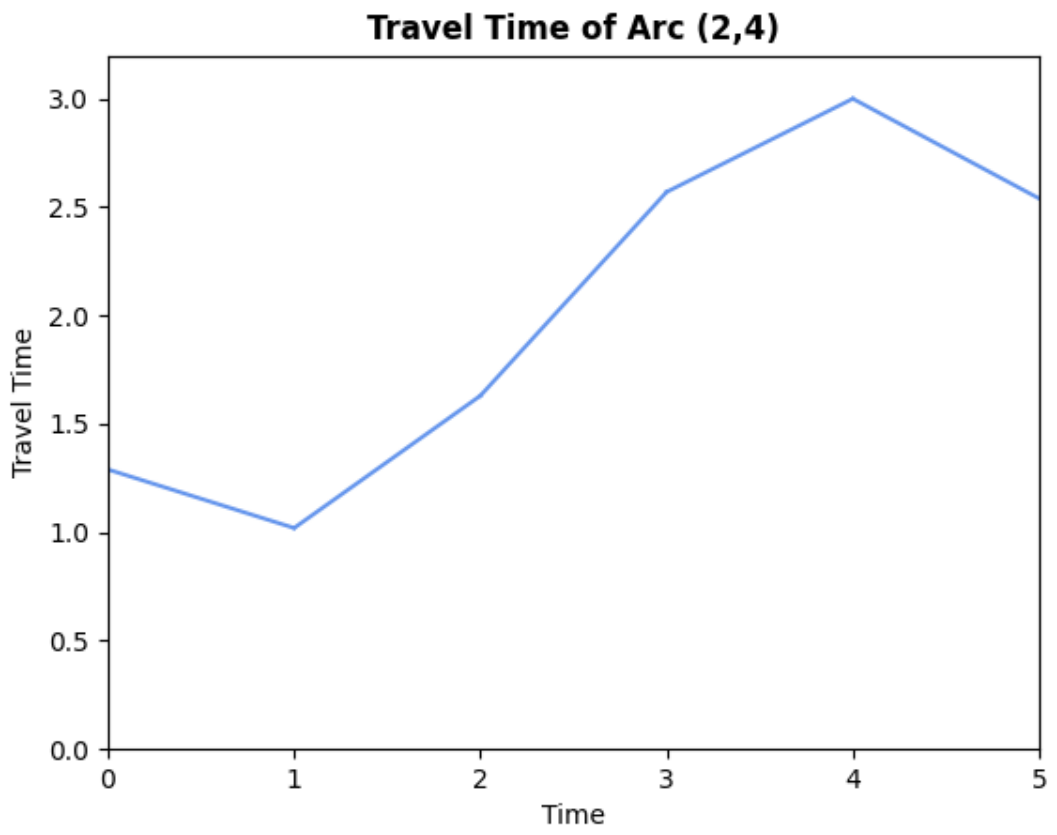
BP	Cung di chuyển				
	(1,2)	(1,3)	(2,3)	(2,4)	(3,4)
0	1.34	2.85	1.99	1.29	0.61
1	0.66	2.95	1.82	1.02	0.73
2	0.14	3.00	1.51	1.63	0.83
3	0.01	2.98	1.10	2.57	—
4	0.35	2.90	0.67	3.00	—
5	1.00	2.76	0.30	2.54	1.00



Cung (1,3)



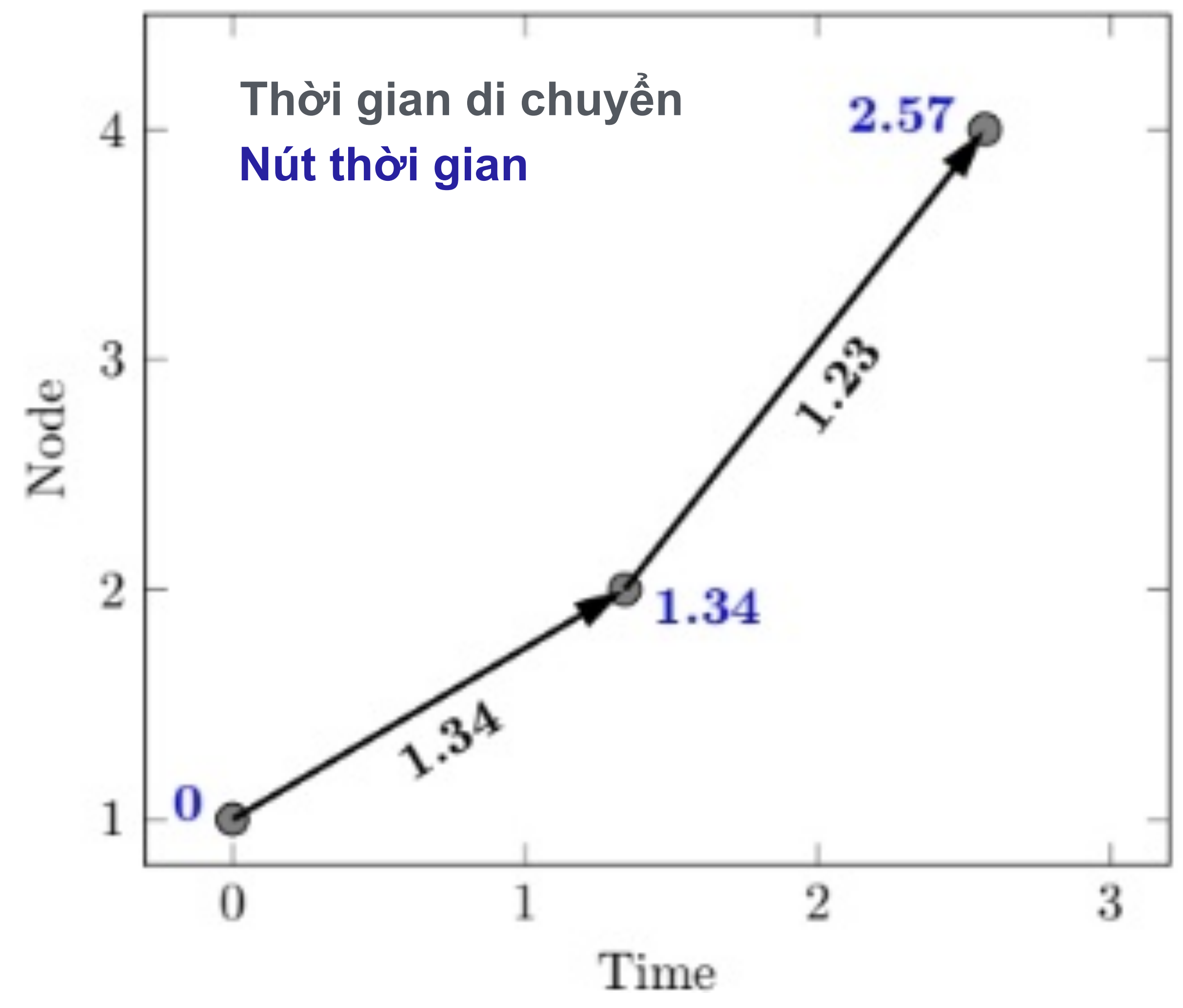
Cung (2,3)



Cung (2,4)

TDSP

Time-dependent shortest path



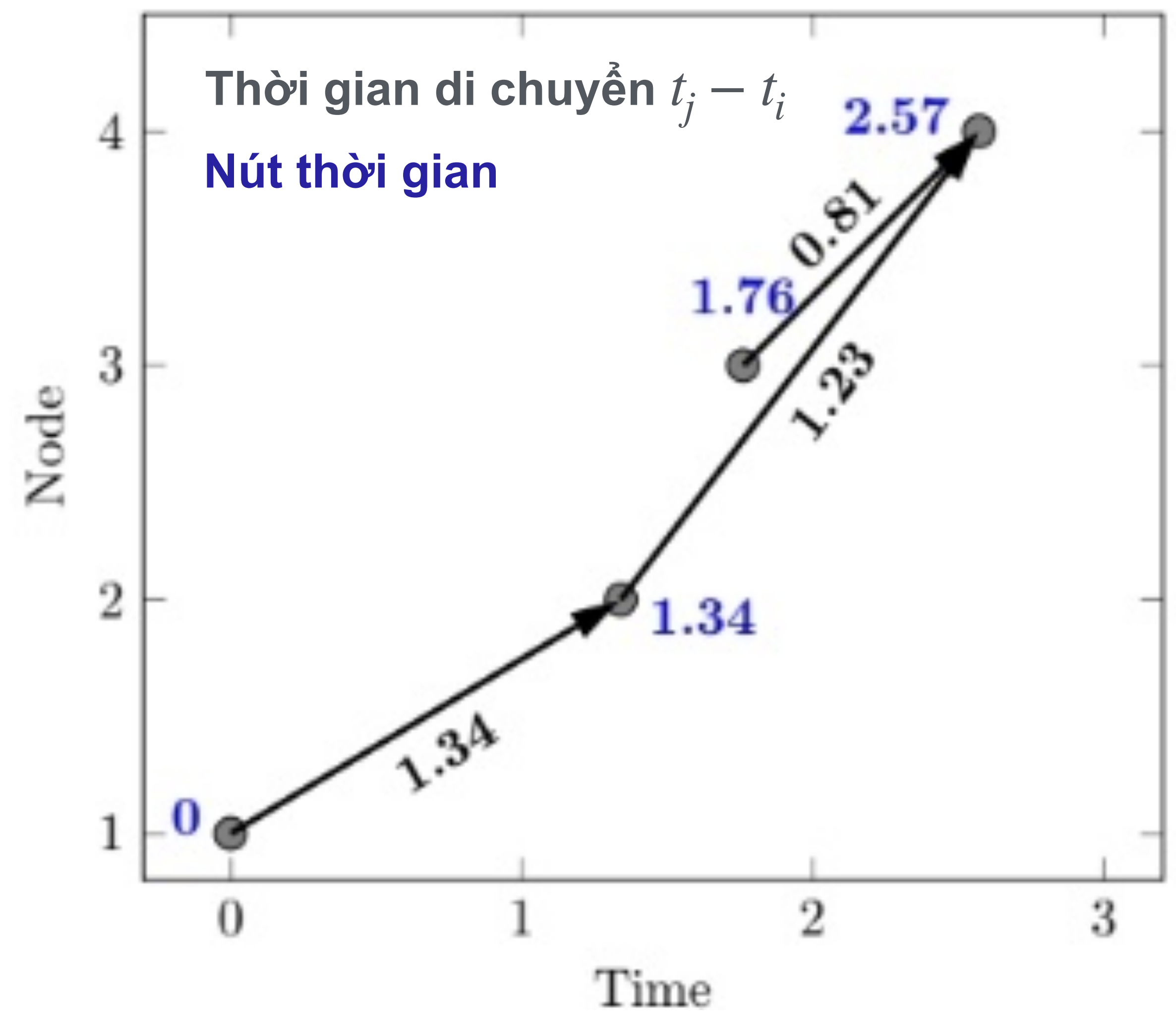
TDSP từ nút (1,0)

- Tìm đường đi ngắn nhất trong mạng phụ thuộc thời gian.
- Sử dụng Djikstra trả về một đường đi => xây dựng lên BSPT
- Ý tưởng cơ bản của Djikstra như sau:
  1. Từ đỉnh nguồn, khởi tạo khoảng cách đến chính nó là 0, khoảng cách đến các đỉnh khác ban đầu là $+\infty$
  2. Chọn đỉnh j có khoảng cách nhỏ nhất trong danh sách và đánh dấu để không đi lại.
  3. Xét lần lượt các đỉnh kề k của đỉnh j, nếu khoảng cách từ nguồn đến k nhỏ hơn
- Độ phức tạp: $\mathcal{O}(SSP)$. (SSP = static shortest path).

BSPT

Backward shortest path tree

- Có kí hiệu là B^t, gốc (n, t), xây dựng từ TDSP trước đó
- Được định nghĩa bởi:
  - tập các nút (i, t_i) với $i \in N$ và $t_i \in (-\infty, T]$
  - tập các cung $((i, t_i), (j, t_j))$ thỏa mãn các tính chất sau:
    - Mỗi $i \in N$ chỉ tồn tại duy nhất t_i là thời gian khởi hành muộn nhất đi từ i đến n
    - Cung $(i, j) \in A$
    - $t_i + c_{i,j}(t_i) = t_j$
    - Chỉ có một đường đi duy nhất từ i đến n trong B^t

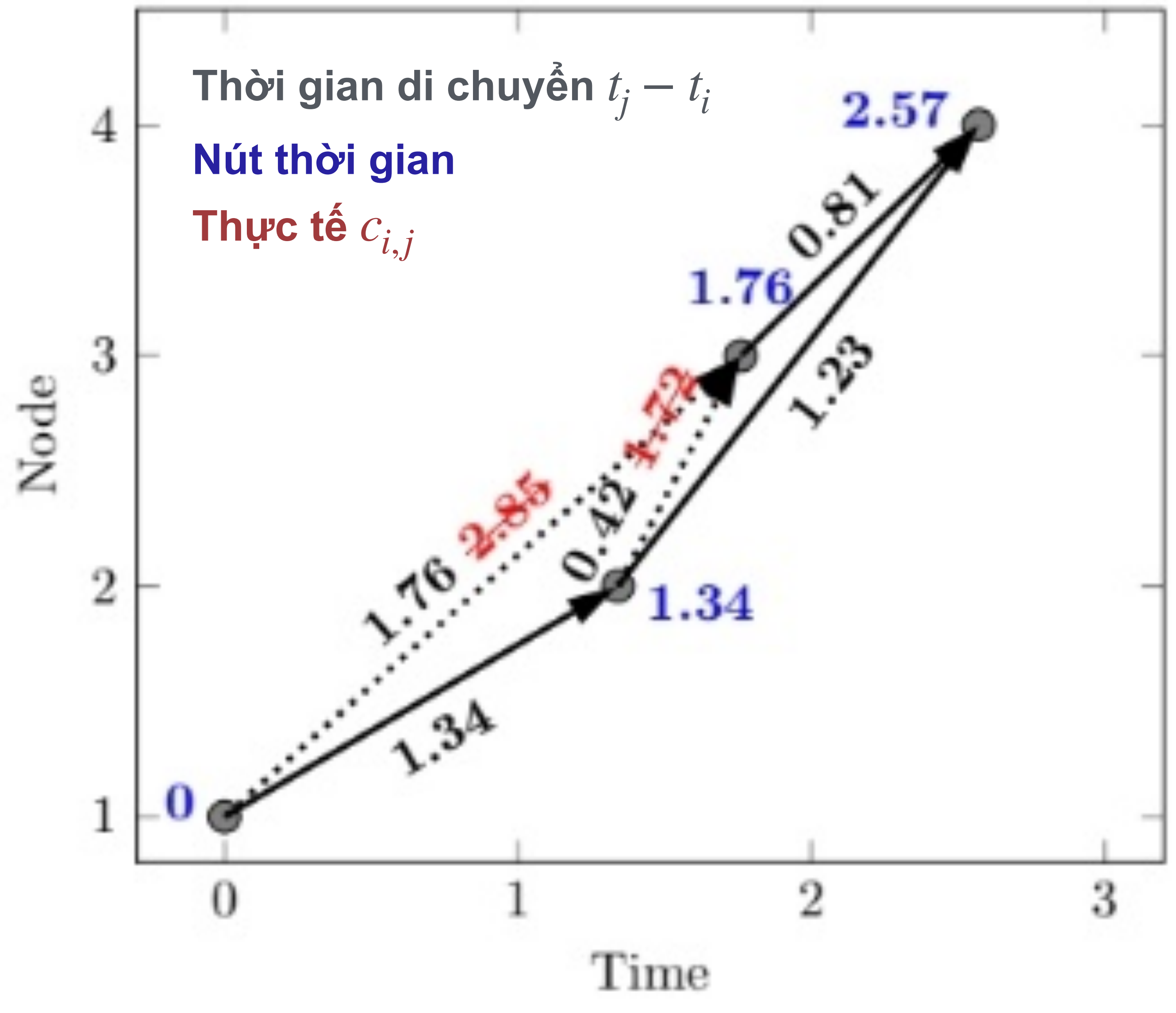


BSPT gốc (4,2.57)

ABSPT

Arc-complete Backward shortest path tree

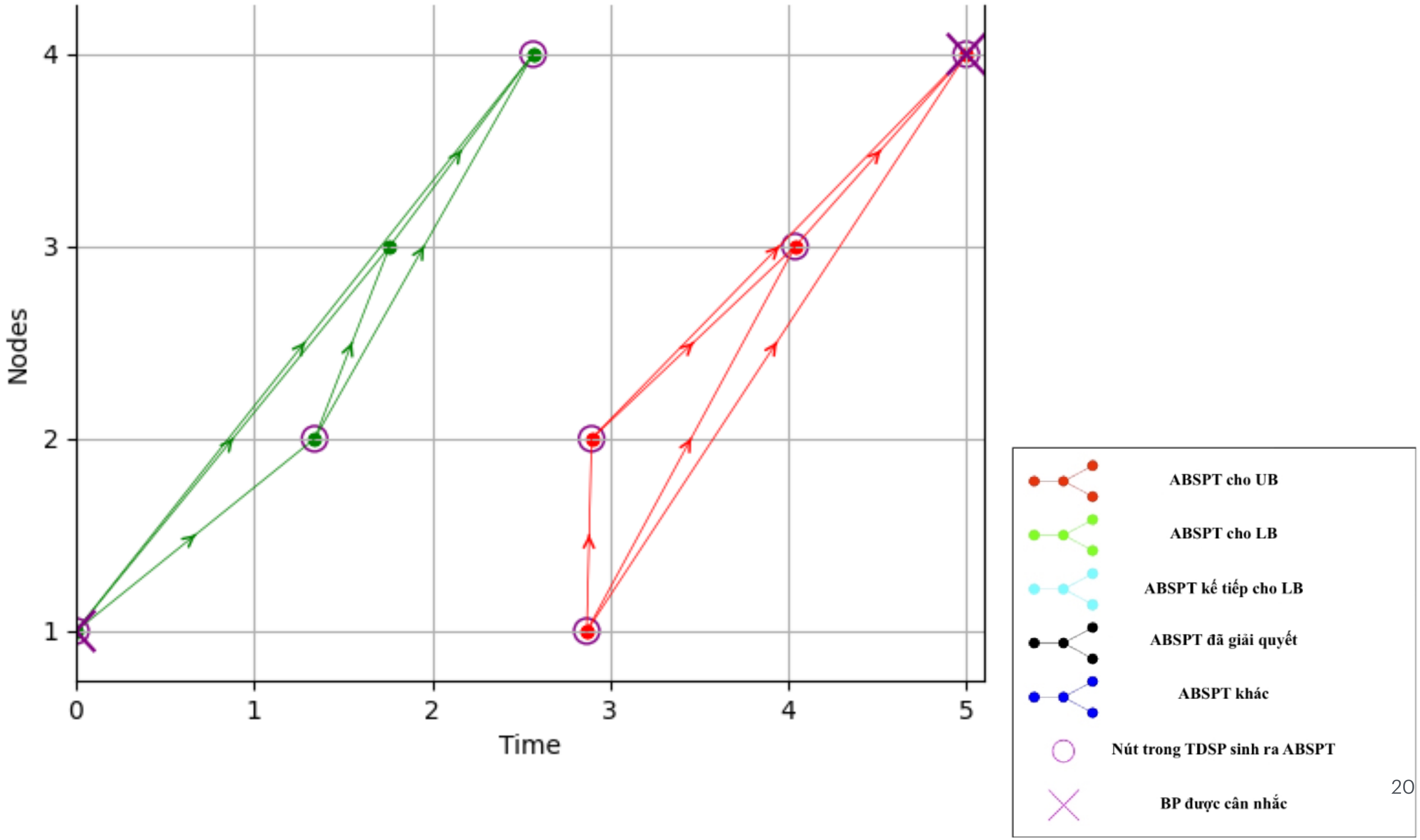
- ABSPT được gọi là BSPT hoàn chỉnh, xây dựng bằng cách thêm các cung vào mỗi nút trong BSPT.
- Tính chất:
  - $t_i + c_{i,j}(t_i) \geq t_j$ cho tất cả các cung trong ABSPT.
- Thuật toán sẽ bắt đầu với hai ABSPT:
  1. ABSPT cho nút $(1,0)$
  2. ABSPT cho nút (n,T)



ABSPT

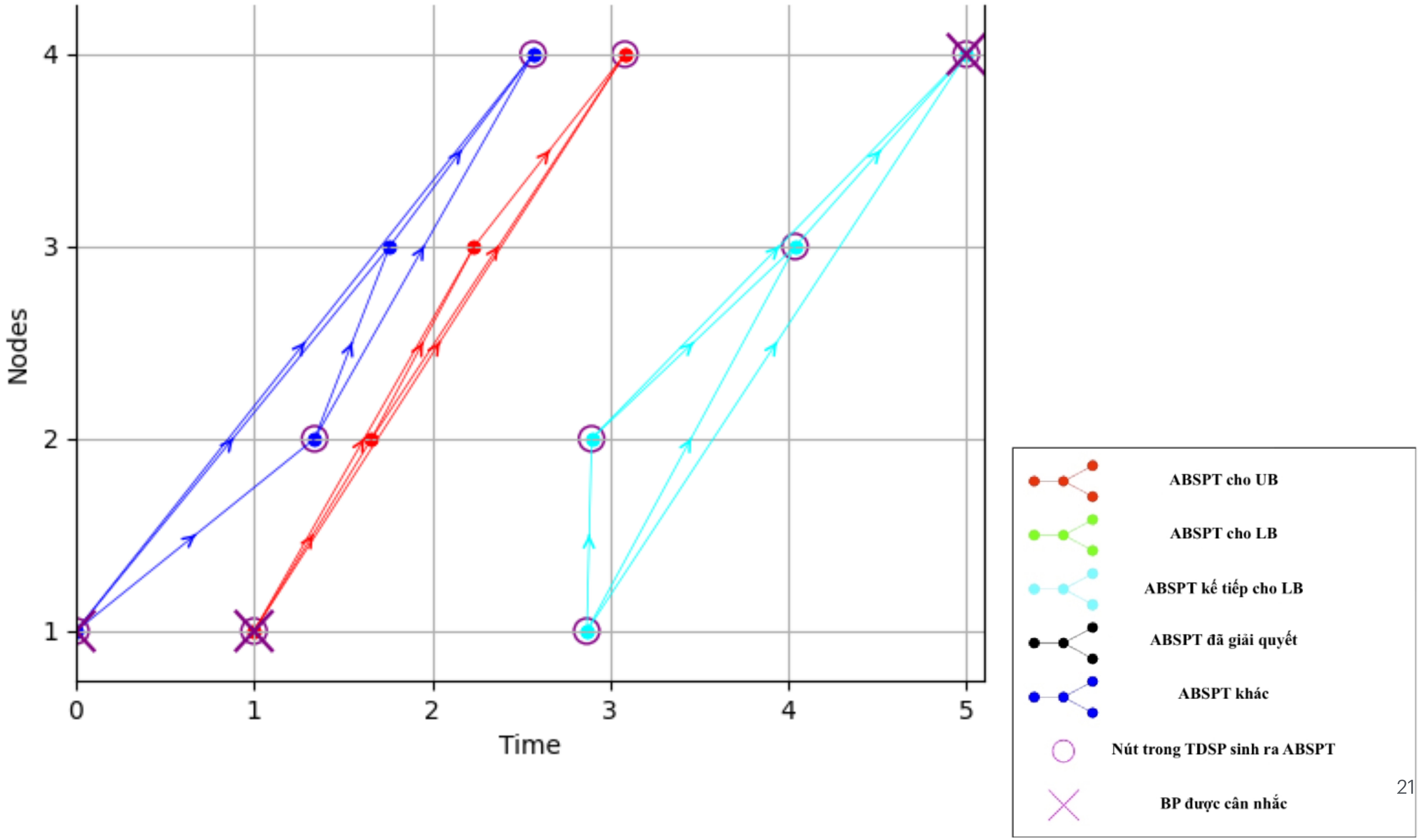
Minh họa thuật toán

Lặp 1



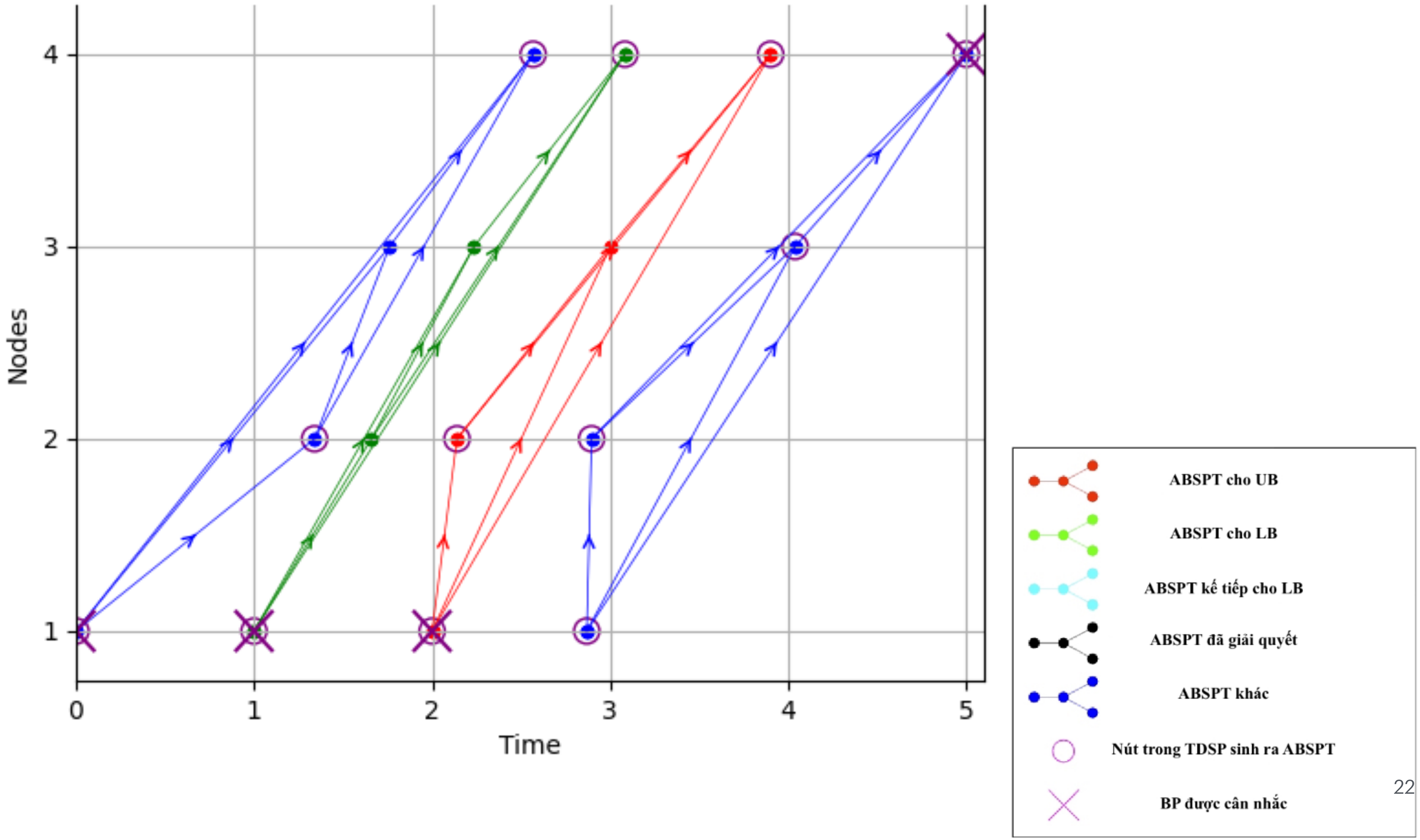
Minh họa thuật toán

Lặp 2



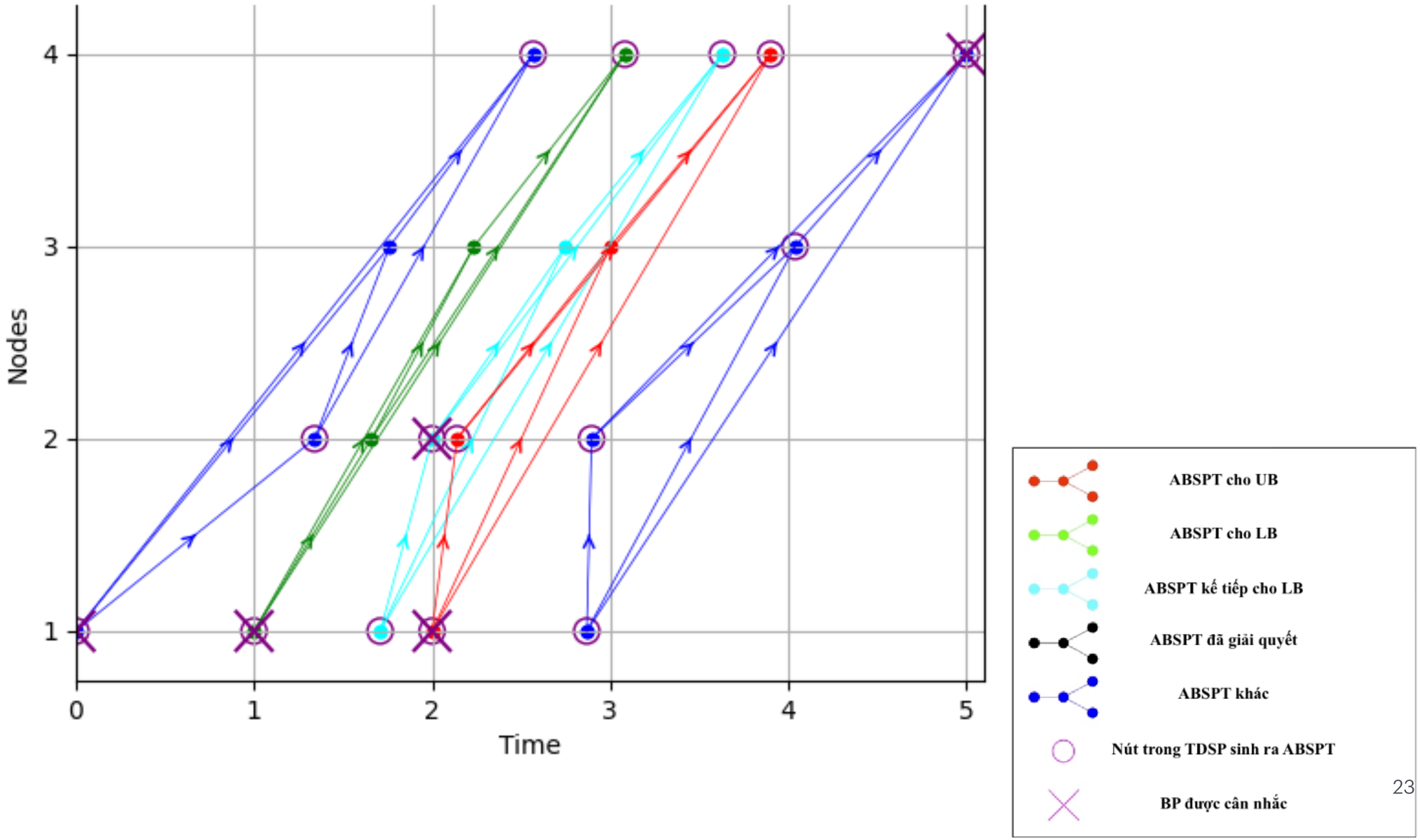
Minh họa thuật toán

Lặp 3



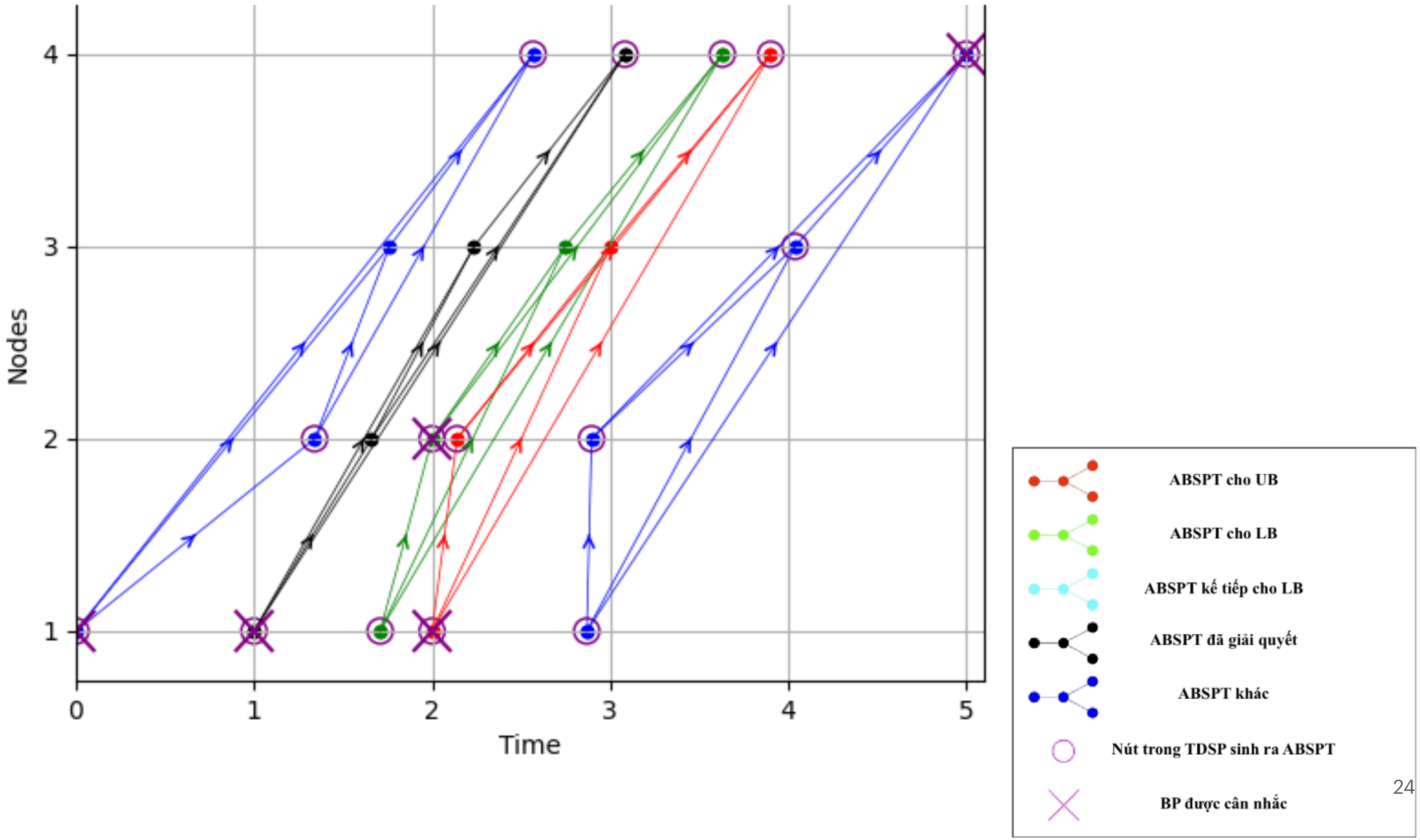
Minh họa thuật toán

Lặp 4



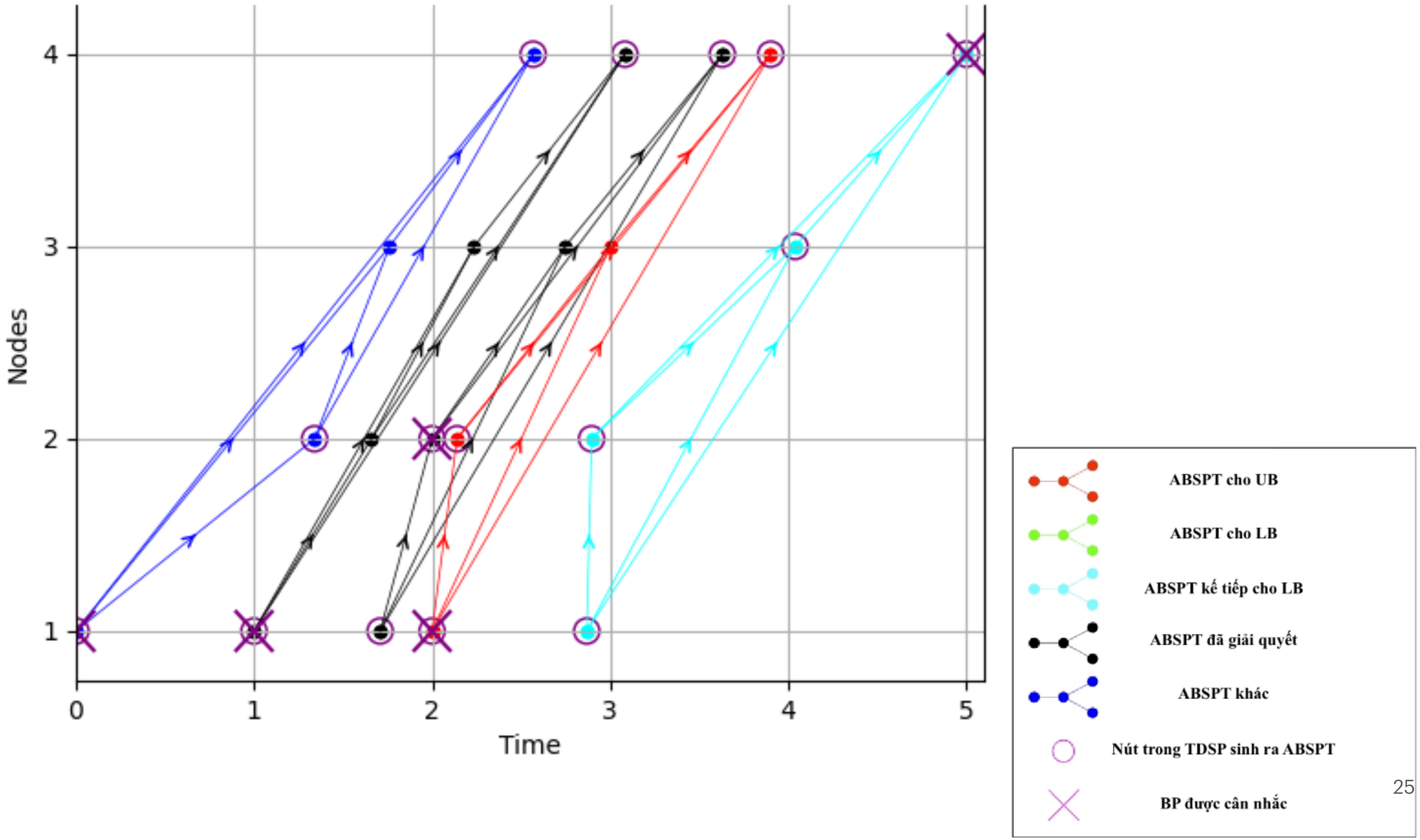
Minh họa thuật toán

Lặp 5



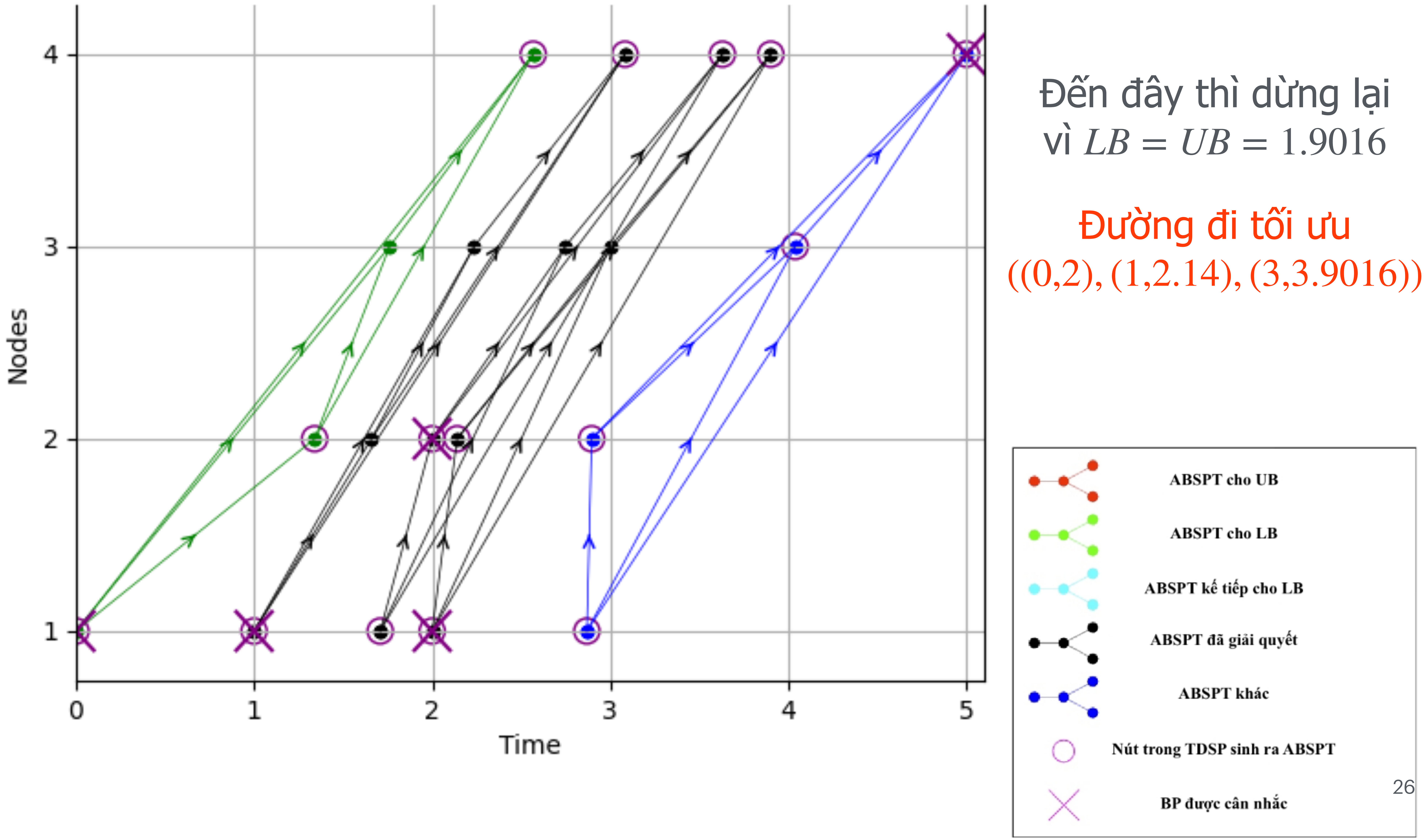
Minh họa thuật toán

Lặp 6



Minh họa thuật toán

Lặp 7



Đến đây thì dừng lại vì $LB = UB = 1.9016$

Đường đi tối ưu
((0,2), (1,2.14), (3,3.9016))

Kết quả thử nghiệm

- Tập dữ liệu:
  - Các kiểu mạng
  - Các hàm thời gian di chuyển
- So sánh với phương pháp Enum

Tập dữ liệu

Các kiểu mạng, mật độ mạng = $\frac{|E|}{|N|^2}$

Kiểu 1: $A = \{(i,j) \in N \times N \,|\, i < j\}$

- Mỗi cặp đỉnh đều có cung nối nhau
- Mật độ mạng: 0.5

Kiểu 2: $A = \{(i, i+1) \,|\, i = 1,\ldots,n-1\} \cup \{(i,j) \in N \times N \,|\, i < j+1$ và $U_{i,j} < 0.5\}$

- Gồm nửa số cung trong kiểu 1. Mọi cung giữa hai đỉnh liên tiếp được giữ lại.
- Mật độ mạng: 0.25

Kiểu 3: $A = \{(i,j) \in N \times N \,|\, i < j < i+4\}$

- Mỗi đỉnh nối với ba đỉnh kế tiếp
- Mật độ mạng: $\sim 3/n$

Tập dữ liệu

Hàm thời gian di chuyển

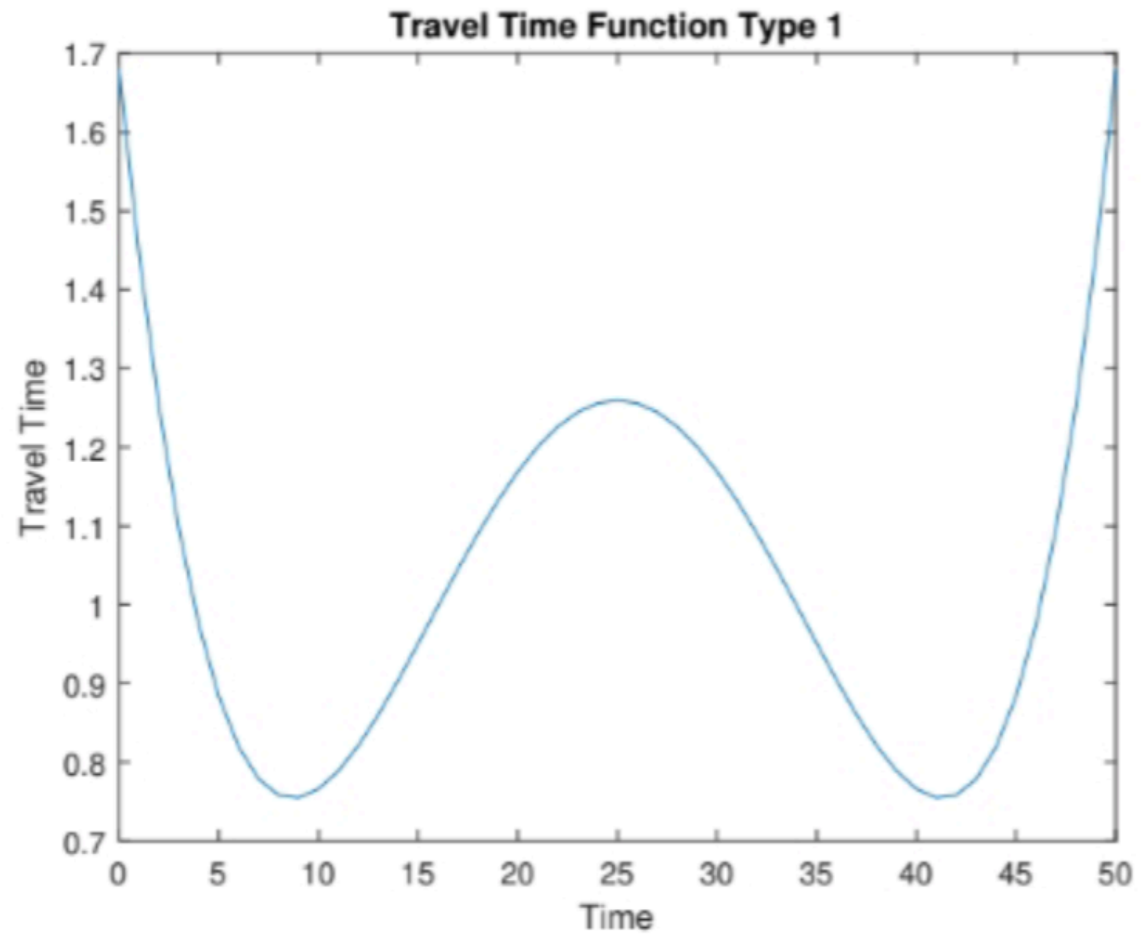
Loại 1: f là đa thức bậc 4:

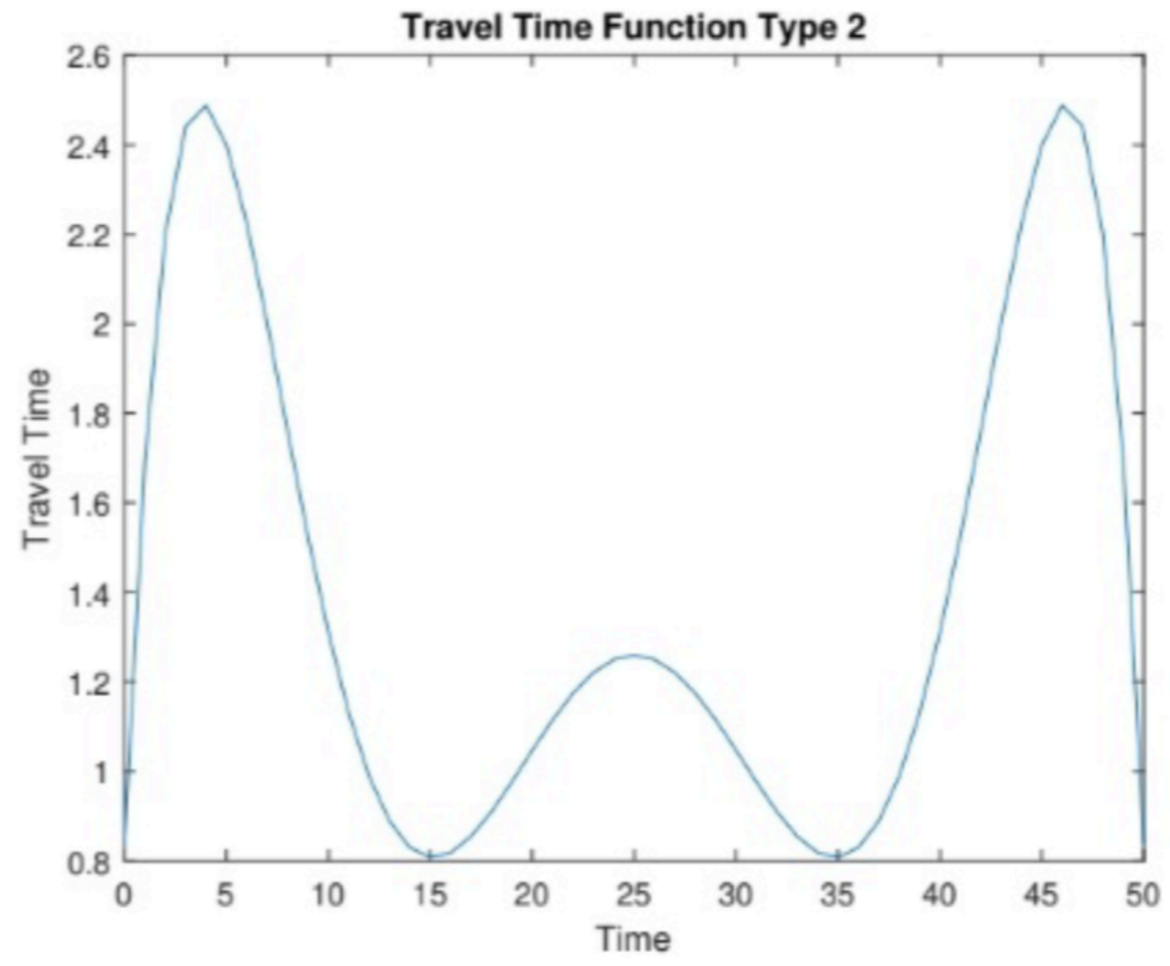
$$f([0, \frac{T}{4}, \frac{T}{2}, \frac{3T}{4}, T]) = \begin{cases} [1.6, 1, 1.05, 1, 1.6](B_{i,j}\frac{|j-i|}{10}), & \text{nếu } U_{i,j} < \frac{1}{3}, \\ [2, 1, 1.5, 1, 2](B_{i,j}\frac{|j-i|}{10}), & \text{nếu } \frac{1}{3} \leq U_{i,j} < \frac{2}{3}, \\ [2.5, 1, 1.75, 1, 2.5](B_{i,j}\frac{|j-i|}{10}), & \text{còn lại,} \end{cases}$$

Loại 2: f là đa thức bậc 6:

$$f([0, \frac{T}{6}, \frac{T}{3}, \frac{T}{2}, \frac{2T}{3}, \frac{5T}{6}, T]) = \begin{cases} [1, 1.6, 1, 1.05, 1, 1.6, 1](B_{i,j}\frac{|j-i|}{10}), & \text{nếu } U_{i,j} < \frac{1}{3}, \\ [1, 2, 1, 1.5, 1, 2, 1](B_{i,j}\frac{|j-i|}{10}), & \text{nếu } \frac{1}{3} \leq U_{i,j} < \frac{2}{3}, \\ [1, 2.5, 1, 1.75, 1, 2.5, 1](B_{i,j}\frac{|j-i|}{10}), & \text{còn lại,} \end{cases}$$

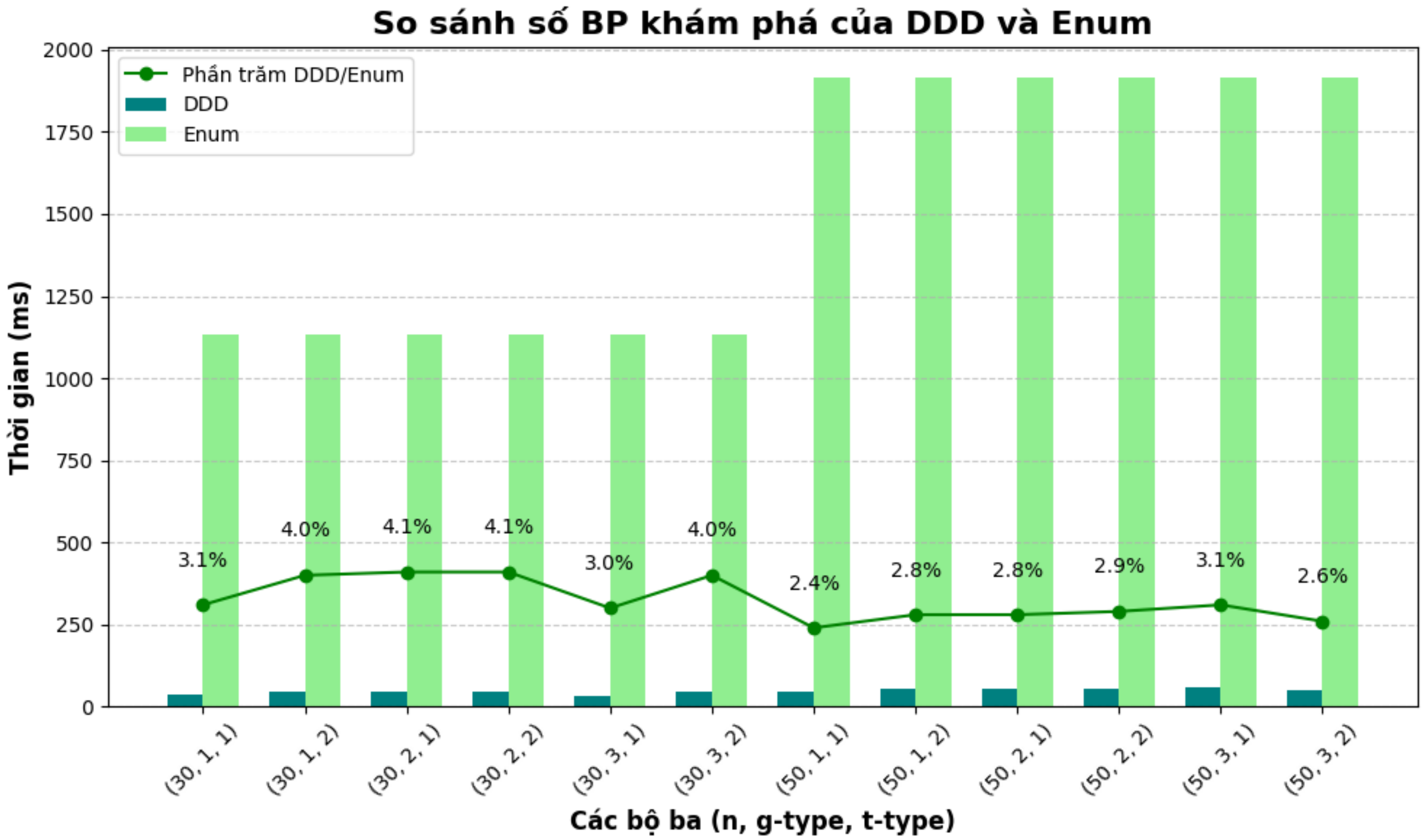
Với $B_{i,j}, U_{i,j} \sim U[0, 1]$.

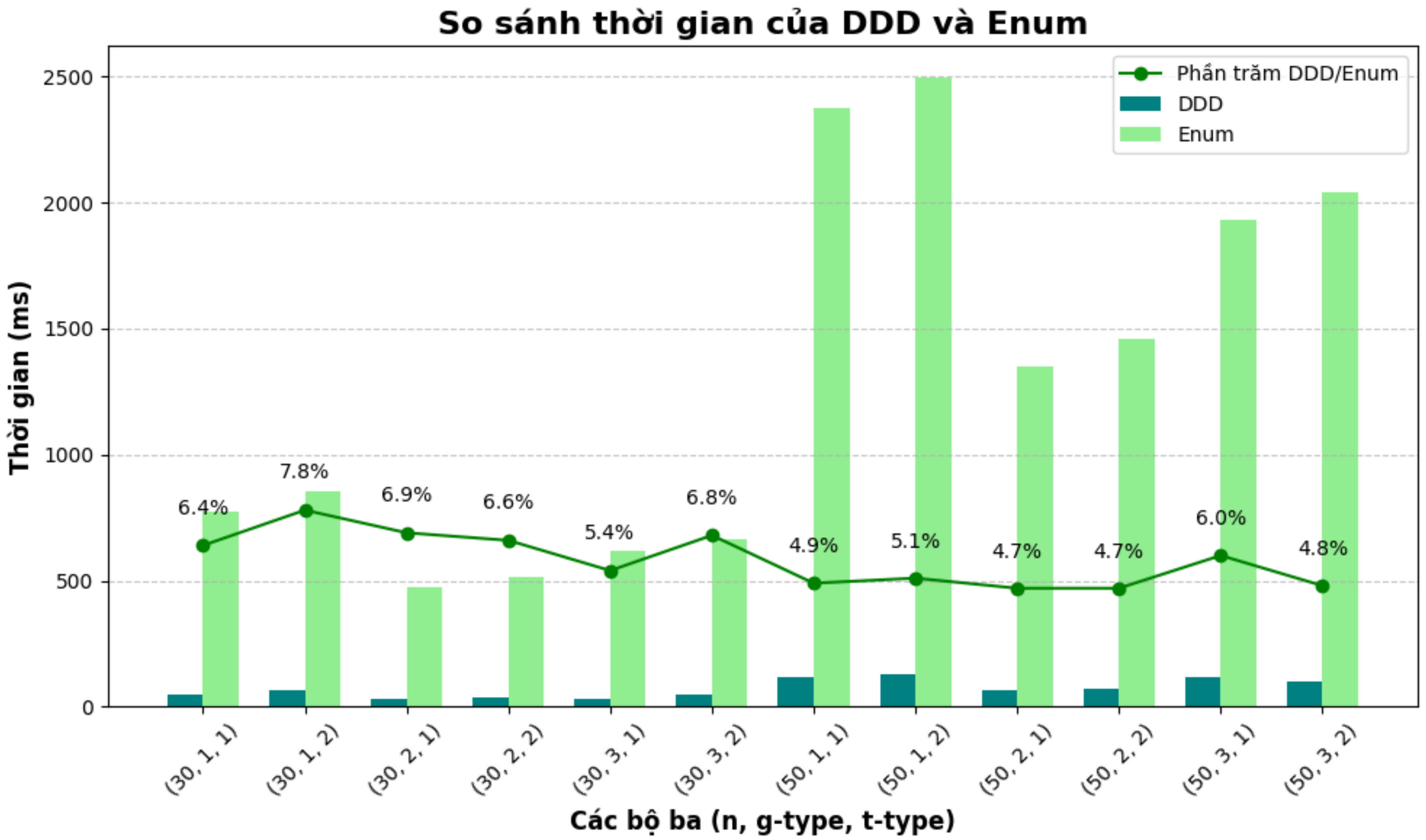




Chạy trên bộ dữ liệu

MDP với n = [30, 50] và T = 40, cách chọn MED





Kết luận

- Khóa luận đã trình bày được thuật toán DDD để giải quyết bài toán MDP
- Khóa luận đã tiến hành thử nghiệm và chứng minh được tính hiệu quả so với phương pháp Enum
- Những hạn chế của khóa luận:
  - Dữ liệu thử nghiệm chưa quá lớn
  - Chưa có dữ liệu thực tế
- Hướng phát triển: Giải quyết bài toán cho phép chờ đợi các nút

Cảm ơn các quý thầy cô đã lắng nghe