Đại học Quốc gia Hà Nội
Trường Đại học Khoa học Tự nhiên
Khoa Toán - Cơ - Tin học

Bùi Khánh Duy

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

Khóa luận tốt nghiệp
Ngành: Khoa học máy tính và thông tin

Hà Nội - 2024

Đại học Quốc gia Hà Nội
Trường Đại học Khoa học Tự nhiên
Khoa Toán - Cơ - Tin học

Bùi Khánh Duy

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

Khóa luận tốt nghiệp
Ngành: Khoa học máy tính và thông tin

Cán bộ hướng dẫn: TS. Vũ Đức Minh

Hà Nội - 2024

Lời mở đầu

Tui xin tự cảm ơn tui vì lười gõ latex mà lại chăm ngồi code latex để đẻ ra con này

Hà Nội, ngày 24 tháng 4 năm 2024

Bùi Khánh Duy

Bùi Khánh Duy

Mục lục

Danh sách hình vẽ

Danh sách bảng

1 Giới thiệu

Bài toán tìm đường đi có thời gian đến nhỏ nhất (MATP). Phương pháp đầu tiên được phát triển để giải quyết vấn đề này dựa trên thuật toán Bellman-Ford ([1], [7]) và được mô tả bởi [3]. Tổng quan về các phương pháp khác cho biến thể này được đề cập bởi [4].

MDP được nghiên cứu trong bối cảnh mạng lưới giao thông ([5]), thậm chí trong phân tích mạng xã hội ([9]). MTTP đặc biệt quan trọng trong việc tính toán giảm khí thải trong môi trường đô thị, vì khí thải của một xe hơi trên tuyến đường có thể xấp xỉ theo tỉ lệ với thời gian di chuyển trên tuyến đường đó; phạm vi tốc độ giới hạn trong đô thị (công thức khi thải được đề cập ở [10] với bài toán định tuyến phương tiện phụ thuộc thời gian).

Trong [2] có giới thiệu thuật toán DDD để giải quyết bài toán tối thiểu hóa chi phí cho mạng dịch vụ theo thời gian liên tục, thông qua việc sử dụng các biểu thức quy hoạch nguyên trên mạng thời gian. Giải pháp ở đây là tìm ra các **thời điểm thích hợp nhất**, cho chúng đi qua các hàm tính toán theo tuần tự nhất định để tạo ra mạng thời gian từng phần. Các mạng này được thiết kế để luôn giải được, cho ra một giới hạn kép (cận trên và dưới) cho giá trị tối ưu của thời gian liên tục. Sau khi tìm được tập thích hợp (số ít các thời điểm), mô hình quy hoạch nguyên sẽ cho ra kết quả tối ưu.

Các định nghĩa sử dụng trong các phần sau: - **Cây đường đi ngắn nhất (Shortest-path tree):** Cho đồ thị $G = (V, E)$, một cây gốc s ($s \in V$) trong đó nút v ($v \in V$) thuộc cây thì đường đi từ nút gốc s đến nút v chính là đường đi ngắn nhất.

2 Mô tả bài toán

Cho một đồ thị có hướng $D = (N, A)$, trong đó: $N = 1, 2, ..., n$ là tập đỉnh và $A \subseteq N \times N$ là tập cạnh; khung thời gian $[0, T]$ và các hàm thời gian di chuyển tuyến tính từng khúc $c_{i,j}(t) > 0$ với $t \in [0, T]$ thỏa mãn tính chất FIFO cho mọi cạnh $(i, j) \in A$.
Tính chất FIFO, trong hàm tuyến tính, có nghĩa là **độ dốc của các khúc ít nhất phải là** -1**.** Không mất tính tổng quát, ta luôn đi từ đỉnh nguồn 1 và kết thúc ở đỉnh đích n.

Đường đi $P = ((i_1^P, t_1^P), (i_2^P, t_2^P), ..., (i_{m(P)}^P, t_{m(P)}^P))$ là một dãy gồm $m(P)$ cặp đỉnh - thời gian, trong đó dãy các đỉnh $(i_1^P, i_2^P, ..., i_{m(P)}^P)$ tạo thành đường đi trong đồ thị D từ i_1^P đến $i_{m(P)}^P$, và mỗi nút trong dãy i_k^P tương ứng với thời gian t_k^P, đại diện cho thời gian bắt đầu di chuyển trên cạnh (i_k^P, i_{k+1}^P) với $k = 1, ... m(P) - 1$. Riêng đỉnh i_k^P với $k = m(P)$ biểu diễn thời gian đến đích. Do đó dãy các cặp cần thoả mãn tính chất:

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

với $k = 1, ... m(P) - 1$. Nếu $t_k^P \geq t_{k-1}^P + c_{i_{k-1}^P, i_k^P}(t_{k-1}^P)$ với bất kì $k \geq 2$, (hoặc nếu $t_1^P > 0$), thì ta nói rằng đỉnh i_k^P cần phải đợi.
Nếu $t_k^P = t_{k-1}^P + c_{i_{k-1}^P, i_k^P}(t_{k-1}^P)$ với mọi $k \geq 2$ thì ta nói rằng P không có chờ đợi (waiting-free).
Nếu P bắt đầu tại đỉnh $i_1^P = 1$ với $t_1^P \geq 0$ và kết thúc tại đỉnh $i_{m(P)}^P = n$ với $t_{m(P)}^P \leq T$ thì ta nói P là một đường đi khả thi. Ta kí hiệu $\mathcal{P}$ biểu diễn tập hợp tất cả các đường đi khả thi như vậy.

Ba hàm mục tiêu chính được quan tâm ở đây là:

1. **Tối thiểu thời gian đến đích:**

$$\min_{P \in \mathcal{P}} t_{m(P)}^P$$

2. **Tối thiểu thời gian thực hiện (đi từ nguồn đến đích):**

$$\min_{P \in \mathcal{P}} (t_{m(P)}^P - t_1^P)$$

3. **Tối thiểu tổng thời gian di chuyển trên các cạnh của đường đi từ nguồn đến đích:**

$$\min_{P\in\mathcal{P}} \sum_{k=1}^{m(P)-1} c_{i_k^P, i_{k+1}^P}(t_k^P)$$

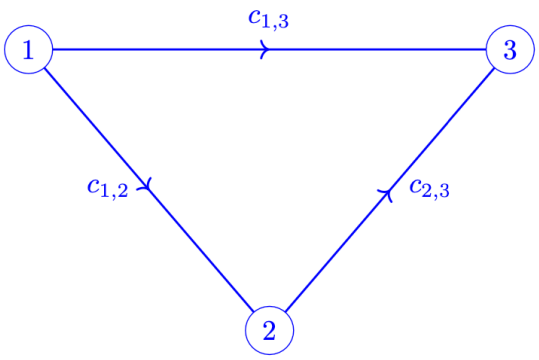
Trong phần còn lại của bài luận, tôi sẽ bỏ đi chữ P ở trên các đỉnh cùng một đường đi để thuận tiện cho ký hiệu và dễ nhìn hơn.

Để minh họa ba hàm mục tiêu này, hãy xem xét đồ thị (a) và các hàm thời gian di chuyển (b,c,d) trong **??**, với $n = 3$ và khung thời gian cho trước là $[0, 4]$.

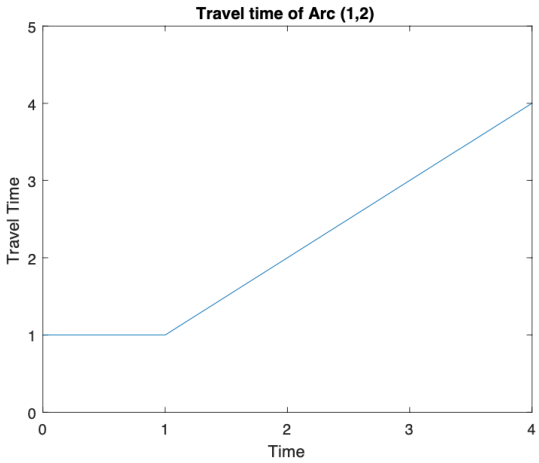
- **Đường đi tối thiểu thời gian đến đích** là $((1, 0), (3, 3.5))$: Xuất phát tại thời điểm $t = 0$, đi qua cạnh $(1, 3)$ và đến đích tại $t = 3.5$, giá trị hàm mục tiêu là 3.5.
- **Đường đi tối thiểu thời gian thực hiện** là $((1, 1.5), (2, 3), (3, 4))$: đường đi bắt đầu từ $t = 1.5$, đi qua cạnh $(1, 2)$ và đến đỉnh 2 tại $t = 3$. Sau đó bắt đầu từ $t = 3$, đi qua cạnh $(2, 3)$ và đến đỉnh đích tại $t = 4$, giá trị mục tiêu là 2.5. Vì không phải chờ đợi trên đường đi này nên tổng thời gian di chuyển cũng là 2.5.
- Tuy nhiên, đường đi trên không cho ra tổng thời gian di chuyển trên các cạnh là tối thiểu. Có nhiều nghiệm tối ưu cho bài toán này. Đối với $t_1 \in [0, 1]$ bất kỳ, đường đi $((1, t_1), (2, 3), (3, 4))$ có thời gian di chuyển tối thiểu: nó sẽ đi qua cạnh $(1, 2)$, bắt đầu từ thời gian t_1 và đến hết cạnh tại $t_1 + 1 \in [1, 2]$, sau đó chờ tại đỉnh 2 cho đến $t = 3$ và đi qua cạnh $(2, 3)$ tại $t = 4$, mang lại giá trị mục tiêu là 2.

Vì các hàm thời gian di chuyển có tính chất FIFO, khi giảm thời gian di chuyển đến đích hoặc thời gian thực hiện, đường đi tối ưu luôn thỏa mãn tính chất $t_k + c_{i_k, i_{k+1}}(t_k) = t_{k+1}$ với $k = 1, \dots, m(P) - 1$. Như đã đề cập ở trên, nghiệm tối ưu sẽ không xuất hiện chờ đợi. Ngoài ra, khi giảm thời gian đến đích, đường đi tối ưu sẽ có $t_1^P = 0$.

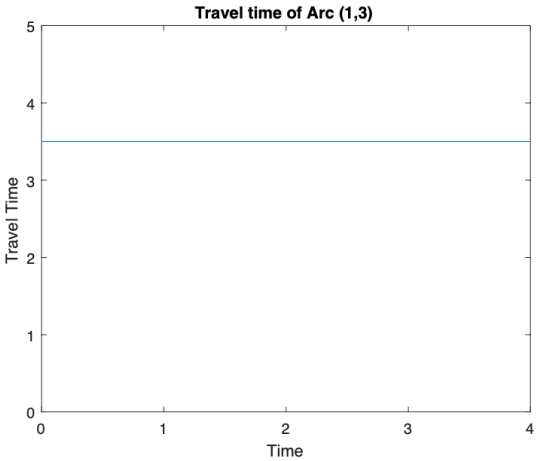
Ba bài báo [3], [11], [4] đã đưa ra phương pháp cho bài toán tìm đường đi đến đích sớm nhất có thể, xuất phát tại đỉnh nguồn với thời gian cố định. Phương pháp này có độ phức tạp đa thức bằng cách mở rộng thuật toán tìm đường đi ngắn nhất tiêu chuẩn. Tương tự áp dụng với bài toán tìm đường đi xuất phát từ nguồn muộn nhất có thể, đến đích với thời gian cố định. Bằng cách tính trước các hàm thời gian di chuyển *đảo chiều*: Với cạnh (i, j) và thời gian t, hàm *đảo chiều* được tính toán tại t sẽ cho ra thời gian di chuyển τ để kết quả thời gian đến đỉnh j là $t - \tau$. Ý tưởng này dẫn đến việc giải bài toán TDSPP với nghiệm là đường đi ngắn nhất TDSP ở các phần sau.



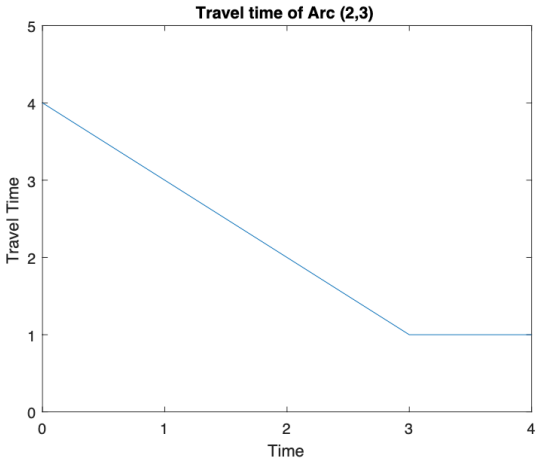
(a) Example Network 1



(b) Travel Time of Arc (1,2)



(c) Travel Time of Arc (1,3)



(d) Travel Time of Arc (2,3)

Figure 1: Example to illustrate the differences between the three objectives.

Hình 2.1: Minh họa các hàm thời gian di chuyển

3 Giải bài toán MDP

Để dễ trình bày, bài luận sẽ chỉ tập trung vào trường hợp thuộc tính *FIFO* ngặt. Với trường hợp *FIFO* không ngặt, chỉ cần vài sự thay đổi nhỏ và chứng minh lại tính đúng đắn là có thể áp dụng.

3.1 Ví dụ

Xét mạng ở hình 2 với các hàm thời gian di chuyển được biểu diễn ở hình 4. Khung thời gian là $[0, 5]$. Hầu hết các cung có điểm dừng là số nguyên. Ngoại lệ là cung $(3, 4)$ chỉ có điểm dừng tại $0, 1, 2$ và 5. **Phụ lục 8** cung cấp chi tiết về các hàm thời gian di chuyển và hàm đảo chiều của chúng.

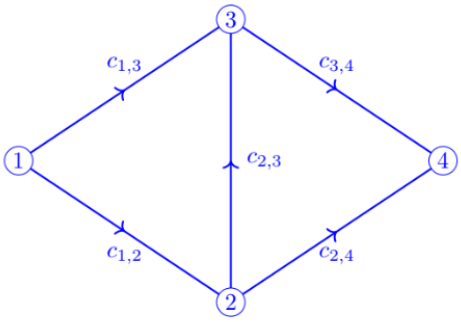


Figure 2: Network D

BP Time	Arc Travel Times				
	(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

Table. 3: Arc Travel Times at Each Breakpoint (BP)

Hình 3.1: Mạng D; Bảng 3: Bảng Thời gian di chuyển với mỗi điểm điểm dừng.

Bảng 3.1: Bảng Thời gian di chuyển với mỗi BP.

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

BP	Cung di chuyển				
	(1,2)	(1,3)	(2,3)	(2,4)	(3,4)
3	0.01	2.98	1.10	2.57	0
4	0.35	2.90	0.67	3.00	0
5	1.00	2.76	0.30	2.54	1.00

3.2 Công thức chi tiết

Cho thời điểm t bất kì thuộc khoảng $[0, T]$ biểu diễn thời gian đi đến đỉnh n, ta có thể xây dựng BSPT tương ứng. BSPT là một Mạng thời gian-không gian (**TEN**) được kí hiệu là B^t, có gốc là (n, t). Cây này được định nghĩa bởi tập hợp các nút[1] (i, t_i) với $i \in N$ và $t_i \in (-\infty, T]$, tập hợp các cung[2] $((i, t_i), (j, t_j))$ thỏa mãn các điều kiện sau:

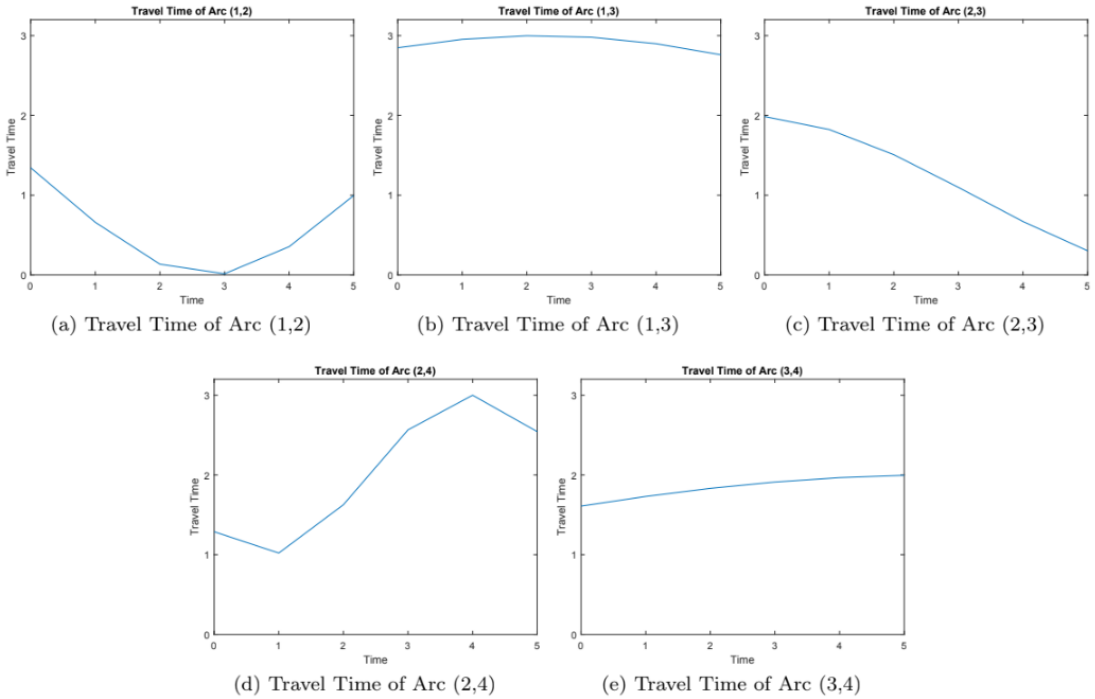
- Với mỗi $i \in N$, chỉ tồn tại t_i là thời gian khởi hành muộn nhất để đi từ i đến n. Lúc này $(i, t_i) \in B^t$,
- $(i, j) \in A$,
- $t_i + c_{i,j}(t_i) = t_j$, và
- **Chỉ có một đường đi duy nhất** từ i đến n trong B^t. Đường đi này được xác định bởi nghiệm của *TDSPP* (được gọi là *TDSP*) bắt đầu từ i tại thời gian t_i và kết thúc ở n.

BSPT được tìm ra bằng cách giải bài toán *TDSPP*, bằng cách áp dụng thuật toán SSSP với một vài điều chỉnh đơn giản. BSPT cho đỉnh 4 tại thời gian $t = 2.57$ được biểu diễn ở hình 5(b). Mạng TEN này bao gồm các nút $\{(1, 0.00), (2, 1.34), (3, 1.76), (4, 2.57)\}$[3] và các cung $(1, 2), (2, 4), (3, 4)$ được xác định bởi các nút theo thời gian.

Lưu ý khi xây dựng BSPT B^t:

- tại một thời điểm $t \in [0, T]$ **bất kỳ,** có thể tồn tại một số đỉnh i không thể đi đến đỉnh n bất kể với thời gian nào > 0.
- **Trường hợp đơn giản:** Nếu đỉnh đang xét i ở thời gian $t = T$, ta có thể loại bỏ nút i khỏi mạng ngay từ bước xử lý trước.

[1] để phân biệt mạng với đồ thị, sử dụng các từ nút thay cho đỉnh.
[2] để phân biệt mạng với đồ thị, sử dụng các từ cung thay cho cạnh.
[3] thời gian làm tròn thời gian đến 2 chữ số thập phân



(a) Travel Time of Arc (1,2) (b) Travel Time of Arc (1,3) (c) Travel Time of Arc (2,3)

(d) Travel Time of Arc (2,4) (e) Travel Time of Arc (3,4)

Figure 4: Arc Travel Times

Hình 3.2: Hình 4: Biểu đồ đường cho Hàm thời gian di chuyển.

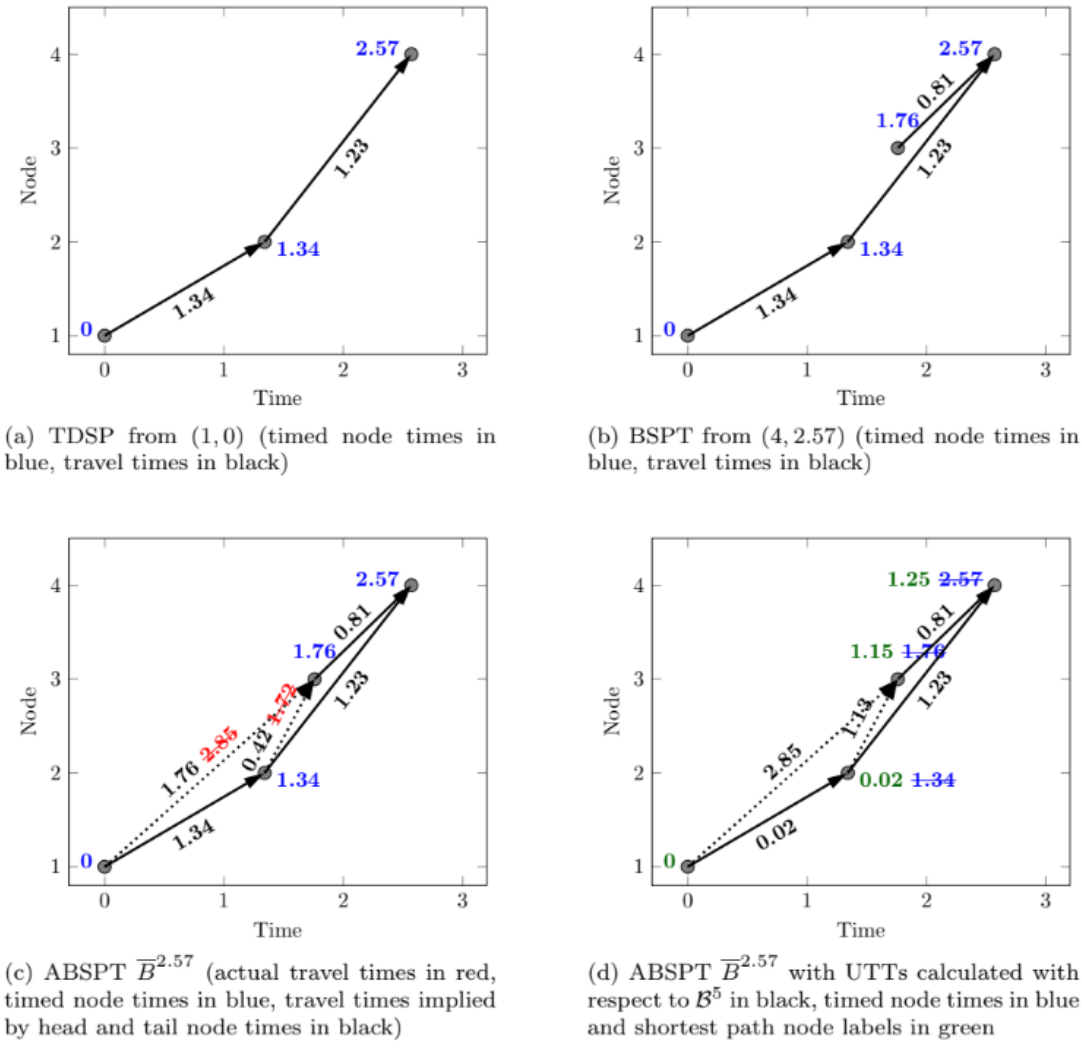
- **Trường hợp phức tạp:** Để đơn giản hóa cho bước tiếp theo, ta giả sử rằng *BSPT* chứa một nút cho mọi đỉnh $i \in N$. Để thực hiện điều này, hàm thời gian di chuyển $c_{i,j}(t)$ được mở rộng sang $t < 0$ (âm): Với mọi cung (i, j) trong mạng A và mọi thời điểm $t < 0$, ta gán giá trị của $c_{i,j}(t) = c_{i,j}(0)$. Do đó, các nút (i, t_i) với $t_i < 0$ cũng có thể được thêm vào mạng B^t.

Thuật toán DDD được xây dựng dựa trên những đặc điểm sau của các BSPT:

- **Tính chất FIFO:** Nếu hai nút (i, s) trong B^t và (i, s') trong $B^{t'}$ với $t' > t$ thì $s' > s$, (Điều này có nghĩa là nếu một nút đến muộn hơn trong B^t, nó cũng sẽ đến càng muộn hơn trong $B^{t'}$).
- **Đường đi theo thời gian tối thiểu:** Bất kỳ đường đi có thời gian tối thiểu nào từ đỉnh 1 đến n, trong đó đến đỉnh n tại thời gian t hoặc muộn hơn, đều được biểu diễn thành một dãy các nút trong B^t. Nghĩa là, với mỗi cặp (i, s) trên đường đi sẽ tồn tại một nút $(i, s') \in B^t$ thỏa mãn $s' \leq s$ (thời gian đến tại s' không trễ hơn s). Lưu ý rằng dãy các nút này không nhất thiết phải theo các cung trong B^t.

Từ những đặc điểm trên, ta có định nghĩa sau:

Definition 1. *Cây **ABSPT** là một mạng **TEN** được hình thành bằng cách thêm*



(a) TDSP from $(1, 0)$ (timed node times in blue, travel times in black)

(b) BSPT from $(4, 2.57)$ (timed node times in blue, travel times in black)

(c) ABSPT $\overline{B}^{2.57}$ (actual travel times in red, timed node times in blue, travel times implied by head and tail node times in black)

(d) ABSPT $\overline{B}^{2.57}$ with UTTs calculated with respect to $\mathcal{B}^5$ in black, timed node times in blue and shortest path node labels in green

Figure 5: Procedure to generate the ABSPT corresponding to $(1, 0)$.

Hình 3.3: Quy trình tạo ABSPT ứng với đỉnh xuất phát (1,0)

*các cung vào **BSPT** có sẵn. Ví dụ, ta thêm cung* $((i,t_i),(j,t_j))$ *cho mọi cặp nút* $(i,j) \in A$ *nếu cả* (i,t_i) *và* (j,t_j) *đều có trong **BSPT** đó. Mỗi ABSPT sẽ tương ứng với một giá trị thời gian là kết thúc của cây.*

Theo **??**, ta có:

$$t_i + c_{i,j}(t_i) \geq t_j$$

cho tất cả các cung trong **ABSPT**. Hay nói cách khác, $t_j - t_i \leq c_{i,j}(t_i)$ với mọi $((i,t_i),(j,t_j))$ trong **ABSPT**

Hình 5(c) minh họa cho ABSPT của B^t với $t = 2.57$. Trong hình, $c_{i,j}(t_i)$ được gạch chéo màu đỏ và thay thế bằng giá trị $t_j - t_i$ trên tất cả các cung mới được thêm vào để tạo thành **ABSPT**. Các cung mới có kí hiệu ba chấm. Ta kí hiệu ABSPT được tạo nên từ B^t là $\overline{\mathcal{B}}^t$.

Thuật toán hoạt động bằng cách sử dụng một danh sách các ABSPT sắp xếp theo thứ tự thời gian tăng dần. Ban đầu có hai ABSPT:

- ABSPT cho **thời gian đến đích sớm nhất có thể**.
- ABSPT cho **thời gian kết thúc của khung thời gian.**

Hai ABSPT này tạo nên khoảng giới hạn của nghiệm khả thi, tức tất cả các đường đi hợp lệ theo khung thời gian đều phải đến đích trong giới hạn này. Thuật toán sẽ liên tục tạo thêm ABSPT để chia nhỏ giới hạn đấy.

Ví dụ với khung thời gian $t \in [0,5]$, hai ABSPT ban đầu là $\overline{\mathcal{B}}^{2.57}$ và $\overline{\mathcal{B}}^5$ trong đó $\overline{\mathcal{B}}^5$ chứa các nút theo thời gian $(1,2.90),(2,2.92),(3,4.05)$ và $(4,5)$. Việc so sánh hai ABSPT liên tiếp dựa trên thời gian, giả sử $\overline{\mathcal{B}}^t$ và $\overline{\mathcal{B}}^{t^+}$ liên tiếp có thời gian kết thúc tương ứng là t và t^+, $t^+ > t$. Đối với mỗi cung $((i,s_i),(j,s_j))$ trong $\overline{\mathcal{B}}^t$, thuật toán tính ra UTT khi ghép với $\overline{\mathcal{B}}^{t^+}$ theo công thức:

$$\underline{c}_{(i,s_i),(j,s_j)} = \min_{\tau}\{c_{ij}(\tau) | s_i \leq \tau \leq s_i^+\},$$

trong đó (i,s_i^+) là nút cho đỉnh i trong ABSPT kế tiếp.

Hình 5(d) minh họa các giá trị **UTT** trong ví dụ với $t = 2.57$ và $t^+ = 5$ (các cung màu đen). Chúng được tính dựa trên đoạn thời gian tạo thành từ các nút trong $\overline{\mathcal{B}}^{2.57}$ và $\overline{\mathcal{B}}^5$. Ví dụ: UTT cho cung $((1,0),(2,1.34))$ là $\min c_{1,2}(\tau) = 0.02$ với $\tau \in [0,2.90]$, và τ nằm ở cuối của đoạn. Ngược lại, UTT cho cung $((1,0),(3,1.76))$ là $\min c_{1,3}(\tau) = 2.85$ với $\tau \in [0,4.05]$ và τ nằm ở đầu đoạn.

Đường đi có chứa UTT nhỏ nhất trong số các ABSPT là **cận dưới** của tất cả nghiệm khả thi. Ví dụ, trong hình 5(d), các nút thuộc đường đi có **UTT** nhỏ nhất trong $\overline{\mathcal{B}}^{2.57}$ được tô màu xanh. Nhãn của nút cuối là 1.25, nghĩa là không có đường dẫn hợp lệ nào đến nút cuối trong khung thời gian từ 2.57 đến 5 có thời gian di chuyển nhỏ hơn 1.25, tương ứng với cận dưới là 1.25.

Bằng việc thêm vào danh sách các ABSPT với thời gian kết thúc lớn hơn ABSPT của cận dưới hiện tại, ta có thể cải thiện (ít nhất là không làm giảm) các giá trị UTT.

Từ ABSPT ta luôn tìm được **cận trên** cho thời gian thực hiện. Theo định nghĩa, nếu $(1, t_1)$ và (n, t_n) là hai nút đầu và cuối tương ứng, thì $t_n - t_1$ sẽ là cận trên và thuật toán sẽ lưu lại cận trên tốt nhất. Trong ví dụ: $\overline{\mathcal{B}}^{2.57}$ có thời gian thực hiện là 2.57 và $\overline{\mathcal{B}}^{5}$ là 2.10. Từ đây ta có cận dưới tốt nhất là 1.25 và cận trên tốt nhất là 2.10.

Bằng việc áp dụng phương pháp tìm cận dưới và cận trên cùng với thêm các ABSPT mới ngay sau ABSPT cận dưới vào danh sách, thuật toán sẽ luôn hội tụ. Thêm vào đó, [8] đã chỉ ra rằng luôn tồn tại một phương pháp tối ưu sử dụng các BP của hàm thời gian di chuyển. Từ đây ta có thể tạo ra các ABSPT mới một cách nhanh chóng hơn đồng thời loại bỏ các khoảng thời gian phụ không cần thiết. Cũng vì vậy mà thuật toán sẽ dừng lại sau hữu hạn các lần lặp.

Bây giờ sẽ là phần trình bày cách sử dụng các BP trong hàm thời gian di chuyển để xây dựng thuật toán.

3.3 Điểm dừng

Đầu tiên, khởi tạo các ABSPT chứa ít nhất một nút (i, t) là BP. Cụ thể: $(1, 0)$ nằm trong ABSPT thứ nhất và (n, T) nằm trong ABSPT thứ 2. Các nút $(1, 0$ và (n, T) đều được coi là BP.

Xét hai ABSPT liên tiếp trong danh sách, giả sử ABSPT trước có nút (i, t_i) và ABSPT sau có nút là (i, t_i^+) đồng thời ABSPT trước là cận dưới hiện tại.

Có các trường hợp sau đây sẽ xảy ra:

- **Trường hợp 1:** $\exists (i, j) \in A \mid \tau_i : t_i < \tau_i < t_i^+$ với điểm dừng τ_i.
  Lúc này, ta nói điểm (i, τ_i) nằm giữa hai ABSPT. Xây dựng ABSPT mới như sau:
  1. Tìm TDSP xuất phát từ i thời điểm τ_i đến n.
  2. Giả sử TDSP đi đến n tại thời điểm τ_n. Lúc này BSPT từ τ_n là

$\mathcal{B}^{\tau_n}$ phải có nút (i, τ_i). Các cung được coi là hoàn thành và $\overline{\mathcal{B}}^{\tau_n}$ là ABSPT tương ứng với nút (i, τ_i). Điều kiện $t_n < \tau_n < t_n^+$ phải thoả mãn (theo tính chất của việc tạo danh sách các ABSPT). Lúc này ABSPT tương ứng với (i, τ_i) được chèn vào giữa hai ABSPT có thời gian kết thúc lần lượt là t_n và t_n^+.

- **Trường hợp 2:** $\nexists (i,j) \in A \mid \tau_i : t_i < \tau_i < t_i^+$, không có điểm dừng nằm giữa hai ABSPT.
  Lúc này, ta có thể kết luận trạng thái ABSPT được giải quyết và không có ABSPT mới được thêm vào.

3.4 Thuật toán 1

Tổng quan thuật toán như sau:

1. Xác định một ABSPT bất kì từ danh sách, giả sử là $\overline{\mathcal{B}}^t$ chứa các nút (j, t_j) với mỗi đỉnh j.
2. Tính toán:
   1. Hàm cận trên $computeUB(\overline{\mathcal{B}}^t) = t_n - t_1$.
   2. Hàm cận dưới $computeLB(\overline{\mathcal{B}}^t)$:
      1. Xây dựng tập các UTT cho mỗi cung trong $\overline{\mathcal{B}}^t$ kết hợp với ABSPT kế tiếp,
      2. Tìm đường đi UTT nhỏ nhất từ nút $(1, t_1)$ đến (n, t_n) trong ABSPT,
      3. Trả về giá trị tìm được.
   3. Lưu lại các giá trị $LB^t \leftarrow computeLB(\overline{\mathcal{B}}^t), UB^t \leftarrow computeUB(\overline{\mathcal{B}}^t)$
3. Đối với ABSPT cuối cùng trong danh sách: $\overline{\mathcal{B}}^T$:
   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 $LB^T = UB^T$ (bằng cận trên).

Vì **??** thực chất là một họ các thuật toán nên không đề cập đến phương án chọn điểm dừng khi có nhiều lựa chọn. **??** sẽ đề cập đến các cách chọn được sử dụng.

3.5 Mã giả

Algorithm 1 Dynamic Discretization Discovery (DDD) Algorithm for the MDP

Input: digraph $D = (N, A)$, latest time T, arc travel time function $c_{i,j}(t)$ for all $t \in [0, T]$, each $(i, j) \in A$

Output: minimum duration path from node 1 to n departing and arriving at times in $[0, T]$

Solve the TDSP starting from $(1, 0)$ to determine t_0, the earliest time that n can be reached ;

Initialize ordered list of ABSPTs: set $L \leftarrow (\overline{B}^{t_0}, \overline{B}^{T})$;

$UB \leftarrow \min\{computeUB(\overline{B}^{t_0}), computeUB(\overline{B}^{T})\}$;

$LB^{t_0} \leftarrow computeLB(\overline{B}^{t_0})$;

Set $LB \leftarrow LB^{t_0}$, set $t \leftarrow t_0$ and set $t^+ \leftarrow T$;

while $(LB < UB)$ **do**

  if some breakpoints (j, τ) lies between $\overline{B}^{t}$ and $\overline{B}^{t^+}$ **then**

    Solve the TDSP starting from (j, τ) to determine, s, the earliest arrival at n ;

    Create the new ABSPT $\overline{B}^{s}$ and set $UB^s \leftarrow computeUB(\overline{B}^{s})$;

    if $UB^s < UB$ **then**

      $UB \leftarrow UB^s$

    end if

    Insert $\overline{B}^{s}$ in the list L between $\overline{B}^{t}$ and $\overline{B}^{t^+}$;

    $LB^t \leftarrow computeLB(\overline{B}^{t})$;

    $LB^s \leftarrow computeLB(\overline{B}^{s})$;

  else

    The status of $\overline{B}^{t}$ is resolved: set $LB^t \leftarrow UB^t$;

  end if

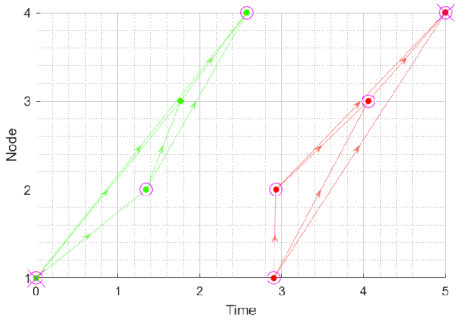
  Update the lower bound: set $t \leftarrow \arg\min_\tau LB^\tau$ and $LB \leftarrow LB^t$

  Identify the next ABSPT in the list: $t^+ \leftarrow min\{\tau : \overline{B}^{\tau}$ is in L and $\tau > t\}$;

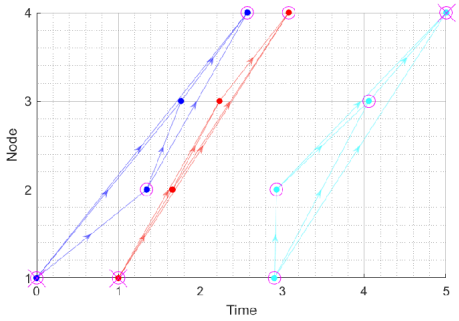
end while

3.6 Minh hoạ thuật toán

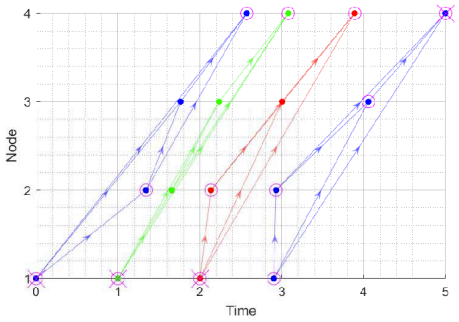
Khởi tạo: Danh sách ABSPT được khởi tạo với hai phần tử:



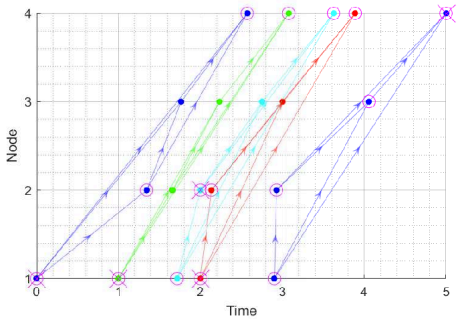
(a) Iteration 1.



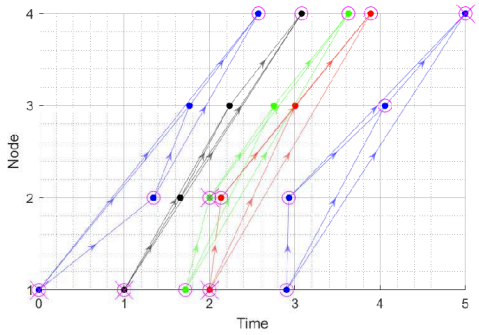
(b) Iteration 2.



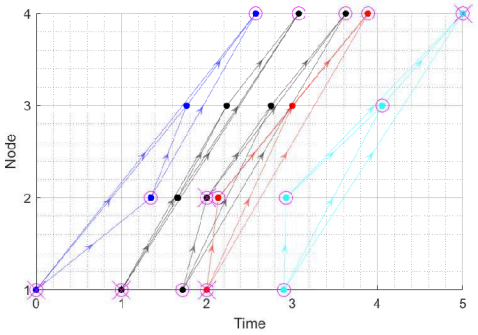
(c) Iteration 3.



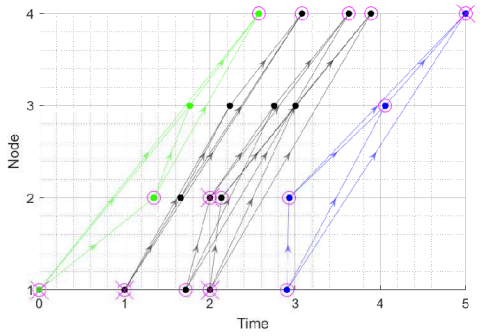
(d) Iteration 4.



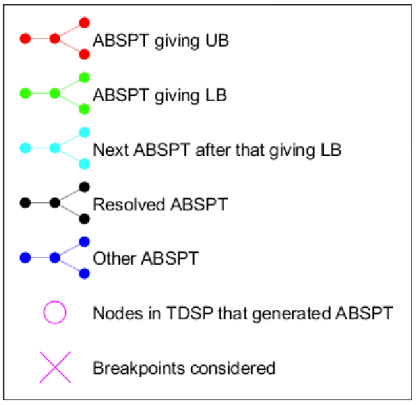
(e) Iteration 5.



(f) Iteration 6.



(g) Iteration 7.



(h) Legend

Hình 3.4: Minh họa cách thức hoạt động của thuật toán trên ví dụ từ **??** và **??**.

- $\overline{\mathcal{B}}^{2.57}$: ứng với nút $(1, 0)$ - TDSP bắt đầu từ $(1, 0)$ đến đỉnh 4 tại thời điểm $t_0 = 2.57$.
- $\overline{\mathcal{B}}^{5}$: như đã đề cập ở trên.

Các giá trị $LB = 1.25$ và $UB = 2.10$ đã được tính toán ở trên.

Lần lặp 1:

1. Xác định điểm dừng $\tau = 1 \in [0, 2.90]$ của cung $(1, 2)$.
2. Tìm thấy điểm dừng mới $(j, \tau) = (1, 1)$ giữa hai **ABSPT**.
3. Tìm thấy TDSP $((1, 1), (2, 1.66), (4, 3.0826))$, đi đến đỉnh 4 tại thời gian 3.08 nên tạo ra $\overline{\mathcal{B}}^{3.08}$ và thêm vào danh sách (Hình 6(b), có 3 ABSPT).
4. Cập nhật cận trên và cận dưới: $UB^{3.08} = 2.08$ và $LB^{3.08} = 1.45$.
5. Tính lại và cập nhật cận dưới cho $\overline{\mathcal{B}}^{2.57}$: $LB^{2.57} = 1.89$.

Hiện tại thì $LB = 1.45$ và $UB = 2.08$. Vì $LB < UB$ nên tiếp tục lặp.

Lặp tiếp theo:

1. Thêm ABSPT $\overline{\mathcal{B}}^{3.89}$ và $\overline{\mathcal{B}}^{3.63}$ tương ứng với $(1, 2)$ và $(2, 2)$. Cái thứ hai cũng là ABSPT thứ ba trong danh sách tại Iteration 4.
2. Cập nhật cận dưới hiện tại: $LB = 1.71$ (từ $\overline{\mathcal{B}}^{3.08}$).
3. Do không có điểm dừng giữa **ABSPT** thứ 2 và thứ 3, cập nhật cận dưới của $\overline{\mathcal{B}}^{3.08}$ bằng cận trên: $LB^{3.08} \leftarrow 2.08$.
4. Cập nhật cận dưới hiện tại: $LB = 1.89$ (từ $\overline{\mathcal{B}}^{3.63}$). (Lặp 5)
5. Cập nhật cận dưới của $\overline{\mathcal{B}}^{3.63}$ và $\overline{\mathcal{B}}^{3.89}$ do không còn điểm dừng.
6. Kiểm tra: $UB = LB = 1.90$. Thuật toán dừng.

Kết quả nghiệm tối ưu (đường đi tối ưu):

$$((1, 2.00), (2, 2.14), (4, 3.89))$$

4 Giải bài toán MTTP

Thuật toán cho MTTP được xây dựng dựa trên ý tưởng của thuật toán cho MDP. Một mạng TEN duy nhất được tạo ra từ các ABSPT trong danh sách L từ **??**. Mạng TEN bao gồm các cung có thời gian liên tiếp $((1,t),(1,t^+)), t < t^+$; mô hình hóa việc chờ đợi tại nguồn đến thời gian thích hợp mới xuất phát và chờ đợi tại đích sau khi kết thúc hành trình. UTT cho các cung này được đặt là 0. Cận dưới được cập nhật theo mỗi lần lặp là giá trị cận dưới của đường đi từ nút $(1,0)$ đến nút (n,T). **??** đã tận dụng được các quan sát sau:

1. Tìm kiếm đường đi cho cận dưới trong TEN có thể được chia nhỏ thành tìm với mỗi ABSPT,
2. Việc thêm ABSPT mới chỉ thay đổi tính toán của một ABSPT.

Sự khác biệt giữa MDP và MTTP nằm ở việc trong MTTP, đường đi tối ưu có thể phải chờ đợi ở một đỉnh khác ngoài đỉnh nguồn. Hệ quả là:

- MTTP không thể chia nhỏ thành các ABSPT riêng lẻ như MDP.
- Cần sử dụng TEN mới để tính cận dưới cho MTTP.
- TEN mới bao gồm các cung chờ đợi giữa các nút chung đỉnh.
- TEN mới không dựa trên ABSPT.

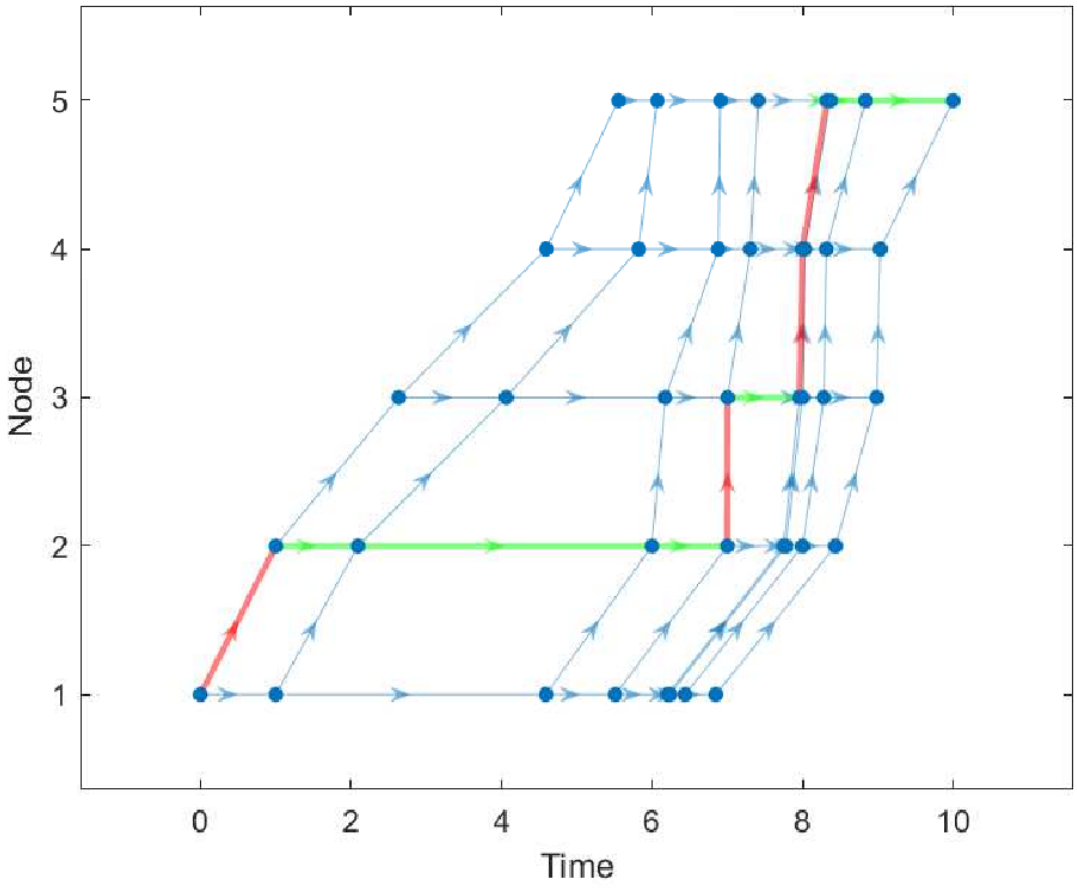
Ta có kết luận: nếu chỉ dùng ABSPT thì không đủ để mô tả giải cho MTTP.

Vì vậy nên phải xây dựng một cấu trúc mới: một TEN mới được tạo ra từ hợp của các FSPT và BSPT từ một đỉnh khác nguồn và đích.

4.1 Nền tảng lý thuyết

Bất kì đường đi khả thi nào cho bài toán MTTP cũng bao gồm các khoảng thời gian di chuyển và chờ đợi, như minh họa trong Hình 7. Thực tế, bất kỳ đường đi nào cũng tương ứng với một danh sách các đường đi không chờ đợi.

Definition 2. *Trong một đường đi nhất định, một đoạn đường con tạo thành từ tối đa các nút con liên tiếp. Giả sử dãy nút đó là* $(i_1,t_1),\ldots,(i_k,t_k)$*, thỏa mãn hai điều kiện: 1.* $t_{l+1} = t_l + c_{i_l,i_{l+1}}(t_l)$ *với mọi* $l = 1,\ldots,k-2$*, và*



Hình 4.1: Illustration of a timed path with 3 periods of waiting in green and 3 travel subpaths in red, in the TEN for a network consisting only of the arcs (1, 2), (2, 3), (3, 4) and (4, 5).

2. $t_k > t_{k-1} + c_{i_{k-1},i_k}(t_k - 1)$ *(phải đợi tại nút* i_k*) hoặc* $t_k = t_{k-1} + c_{i_{k-1},i_k}(t_k - 1)$ *với* $i_k = n$*. Đoạn đường con được định nghĩa là dãy các nút* $(i_1, t_1), ..., (i_{k-1}, t_{k-1}), (i_k, \tau)$*, trong đó* $\tau = t_{k-1} + c_{i_{k-1},i_k}(t_{k-1})$

Xét ví dụ, cho một dãy đường đi như sau:

$$((1, 0); (2, 6.9); (3, 7.9); (4, 8.1); (5, 8.5))$$

với $\tau_{12} = 1$, $\tau_{23} = 0.2$, $\tau_{34} = 0.2$ và $\tau_{45} = 0.4$. Theo **??**, có ba đường con như sau: $\{(1, 0); (2, 1)\}$, $\{(2, 6.9); (3, 7.1)\}$ và $\{(3, 7.9); (4, 8.1); (5, 8.5)\}$. **??** minh hoạ cho ví dụ trên: Mạng TEN với $D = (\{1, 2, 3, 4, 5\}, \{(1, 2), (2, 3), (3, 4), (4, 5)\})$. Đường đi hiển thị bằng màu đỏ và xanh lá cây. Ba đoạn đường con của nó được tô màu đỏ.

Đường con là một phần của đường đi nhưng không có chờ đợi. Nó có các đặc điểm như sau:

- Một đường con ***không thể mở rộng ở hai đầu*** mà vẫn duy trì trạng thái không chờ đợi, hay:
  - Nếu hai nút (i, t_i) và (j, t_j) với i, j liên tiếp và $t_j > t_i + c_{i,j}(t_i)$ thì nút $(j, t_i + c_{i,j}(t_i))$ là kết của đoạn con này và (j, t_j) là nút đầu của đoạn con tiếp theo.
- Một đường đi có thể được tạo thành từ một tập các đường con, với thời gian chờ đợi xen giữa chúng.

Ví dụ: hình dung một người lái xe đi trên đường. Khi đi trên đường cao tốc (đoạn đường con), người lái xe không cần dừng lại. Tuy nhiên, khi đến ngã tư (điểm dừng), người lái xe phải dừng lại và chờ đèn giao thông.

Lemma 1. *Cho một đường đi tối ưu* P *của bài toán* $MTTP$ *trên mạng* D *với hàm thời gian di chuyển* c *và khoảng thời gian* $[0, T]$*. Xét một* ***đường đi con*** S *bất kỳ của* P*. Giả sử* S *bắt đầu tại nút* (i, t_i) *và kết thúc tại nút* (j, t_j)*. Khi đó* S *là nghiệm tối ưu cho bài toán* ***MDP*** *trên cùng mạng* D*, với cùng hàm* c*, nhưng từ nguồn* i *đến đích* j *và khoảng thời gian* $[\tau^-, \tau^+]$*, trong đó:* $\tau^- \leq t_i$ *là thời gian kết thúc của* ***đường đi con*** *ngay trước* S *và* $\tau^+ \geq t_j$ *là thời gian bắt đầu của đường đi con ngay sau S trong P (nếu có). Lưu ý rằng* $\tau^- = 0$ *nếu* $i = 1$ *(không có đường đi con nào trước S) và* $\tau^+ = T$ *nếu* $j = n$ *(tức là không có đường đi con nào sau S).*

Chứng minh. □

Proposition 1. *Với bất kỳ bài toán MTTP nào, đều có một* ***đường đi tối ưu*** *sao cho mỗi* ***đường con*** *của nó đi qua ít nhất một nút tại điểm dừng của hàm thời gian di chuyển.*

Chứng minh. □

Definition 3. *Một MTTP với các hàm thời gian di chuyển c, việc xây dựng mạng thời gian có độ dài của cung, gọi tắt là **TENL**, như sau:*

1. *Đối với mỗi **điểm dừng** (i,t), giải hai bài toán con **TDSPP**: $i-FSPT$ (kí hiệu $\mathcal{F}^{i,t}$) và $i-BSPT$ (kí hiệu $\mathcal{B}^{i,t}$). Tạo ra TEN từ tất cả các nút và cung trong $\mathcal{F}^{i,t}$ và $\mathcal{B}^{i,t}$ với mọi **điểm dừng** (i,t) có trong đó.*
2. *Đối với mỗi nút trong mạng TEN mới tạo, thêm các cung chờ đợi giữa các bản sao thời gian liền kề theo thứ tự thời gian của nút đó trong **TEN**.*
3. *Ngược lại, gán cho bất kỳ cung nào trong **TEN** kết quả, chẳng hạn như $((i,t),(j,t'))$, độ dài bằng 0 nếu $j=i$ và độ dài bằng $c_{i,j}(t)$ (và phải bằng $t'-t$ theo như xây dựng).*

Chưa xong.

Chứng minh. □

Corollary 1. *Cho một **MTTP** có tập các đỉnh $\{1,...,n\}$ với khung thời gian $[0,T]$, **bất kỳ** đường đi ngắn nhất nào từ $(1,0)$ đến (n,T) trong **TENL** đều tương ứng là một nghiệm của bài toán **MTTP**.*

Chứng minh. □

Proposition 2. *Độ phức tạp cho **MTTP** trên đồ thị có hướng (N,A) với $n=|N|$ đỉnh và K^{nodes} điểm dừng là*

$$(\mathcal{K}^{nodes} \times SSP(n,|A|) + ASPP(K^{nodes}n, K^{nodes}n)),$$

*trong đó $SPP(\alpha,\beta)$ là độ phức tạp của việc giải bài toán **đường đi** ngắn nhất tĩnh trên đồ thị có hướng với α nút và β cung, và $ASPP(\alpha,\beta)$ cũng tương tự nhưng đối với một đồ thị có hướng phi chu trình.*

Chứng minh. □

4.2 Công thức chi tiết

Trước khi đi vào chi tiết, hãy nhắc lại sự khác nhau của bài toán MDP và MTTP: MDP sử dụng ABSPT để giảm số điểm dừng cần xử lý; trong khi MTTP có cách tiếp cận khác. Theo Proposition 3.3, cấu trúc của ABSPT

không phù hợp với MTTP. Một nghiệm tối ưu cho MTTP là một chuỗi các đường đi con được nối với nhau bằng cung chờ đợi; mỗi đường đi con được giả định là đi qua một điểm gãy và thời gian chờ đợi luôn dương (> 0).

Definition 4. *Một đường đi con qua điểm dừng (i, t), là một đường đi không chờ đợi nối một TDSP kết thúc tại (i, t) với một TDSP khác xuất phát từ (i, t). Thuật ngữ **đường đi con** sẽ không chỉ định điểm dừng, trong khi **đường đi con** i được dùng để chỉ điểm dừng tại đỉnh i vào thời điểm không xác định.*

Qua định nghĩa, luôn tồn tại một số nghiệm tối ưu cho MTTP mà mỗi **đường đi con** của nó nằm trong một **TEN** được hình thành bởi hợp FSPT và BSPT tại một **điểm dừng** xác định. Do đó, thay vì dùng ABSPT, cấu trúc phù hợp hơn sử dụng để giải cho MTTP là hợp của $i - FSPT$ và $i - BSPT$ cho cùng một điểm dừng tại đỉnh i, gọi là $mangrove$. Lưu ý toán tử hợp ($\cup$) cũng được dùng để tạo một TEN dưới dạng hợp của hai TEN.

Definition 5. *Một $mangrove$ cho điểm dừng (i, t), ký hiệu là $\mathcal{M}^{i,t}$, là một TEN được tạo bằng cách lấy hợp của $\mathcal{F}^{i,t}$ và $\mathcal{B}^{i,t}$, hay $\mathcal{M}^{i,t} = \mathcal{F}^{i,t} \cup \mathcal{B}^{i,t}$. Tôi gọi bất kì $mangrove$ nào như vậy là $i - mangrove$ và sử dụng $(i, t) - mangrove$ nếu chúng tôi muốn chỉ định cho điểm dừng (i, t).*

Giống với khi tính toán BSPT cho MDP, tính chất FIFO phải được đảm bảo cho bất kì đỉnh i nào. Tập tất cả các $i - mangrove$ luôn được sắp xếp theo thời gian đối với các nút trong $i-FSPT$ và tương tự với $i-BSPT$ của chúng. Cụ thể, nếu $\mathcal{M}^{i,t} = F^{i,t} \cup B^{i,t}$ và $\mathcal{M}^{i,t'} = F^{i,t'} \cup B^{i,t'}$ với ($t' > t$), thì với bất kì nút j nào có $(j, s) \in \mathcal{F}^{i,t}$ và $(j, s') \in \mathcal{F}^{i,t'}$ thì $s' > s$. Tương tự với các nút trong $i - BSPT$.

Do đó, bất kỳ đường đi con i nào đến đỉnh i tại thời gian t hoặc muộn tạo thành nút (i, t) hơn đều nằm trong một dãy các nút thuộc $\mathcal{M}^{i,t}$. Đương nhiên điều kiện là phải có các cung nối giữa chúng. Đối với điểm gãy (i, t) đã cho, cây AFSPT (kí hiệu $\overline{\mathcal{F}}^{i,t}$) được tạo từ $\mathcal{F}^{i,t}$ sau khi thêm các cung $((j, t'), (k, t''))$ chưa tồn tại, sao cho $\forall (j, k) \in A$ thì cả (j, t') và (k, t'') đều nằm trong $FSPT$. Tương tự ta có định nghĩa của ABSPT như đã đề cập ở chương về MDP. $Mangrove$ hoàn chỉnh cho điểm dừng (i, t), kí hiệu ($\overline{\mathcal{M}}^{i,t}$) và là một TEN được tạo ra từ hợp của $\overline{\mathcal{F}}^{i,t}$ và $\overline{\mathcal{B}}^{i,t}$.

Thuật toán được thực hiện bằng cách duy trì một tập các TEN, mỗi TEN là một $mangrove$ hoặc $mangrove$ hoàn chỉnh cho một số điểm dừng. Mỗi tập TEN con có chung đỉnh này được sắp xếp theo thời gian của các điểm dừng. Với mỗi đỉnh i tương ứng có một danh sách L^i chứa các điểm dừng (i, t) sắp xếp theo thời gian. Với mỗi (i, t) được thêm vào L^i thì sẽ tính toán $M^{i,t}$ và

$\overline{\mathcal{M}}^{i,t}$. Với mỗi điểm dừng liên tiếp trong L^i, ví dụ như (i,t) và (i,t^+) với $t^+ > t$, ta tiến hành như sau:

- Trường hợp giữa t và t^+ không có điểm dừng:
  Ta nói $mangrove$ cho (i,t) đã được giải quyết. Đặt UTT thành thời gian di chuyển thực tế:

$$\underline{c}_{(i,s),(k,s')} := c_{j,k}(s)$$

  với mỗi cung $((j,s),(k,s'))$ trong $\mathcal{M}^{i,t}$.
- Trường hợp còn lại khi $mangrove$ cho (i,t) chưa được giải quyết:
  Thuật toán đặt UTT trên mỗi cung $((j,s),(k,s'))$ trong $\mathcal{M}^{i,t}$

$$\underline{c}_{(i,s),(k,s')} := \min_{\tau}\{c_{i,k}(\tau) : s \le \tau \le s^+\}$$

  với s^+ là thời gian duy nhất để nút (j,s^+) nằm trong $\overline{\mathcal{F}}^{i,t}$ (tương tự với $\overline{\mathcal{B}}^{i,t}$). Lúc này, bất kỳ đường con tại i trong khoảng thời gian $[t,t^+]$ xuất hiện trong đường đi của $\overline{\mathcal{M}}^{i,t}$ thì đều có tổng UTT không lớn hơn thời gian di chuyển của đường đi đó.

Để đảm bảo rằng L^i luôn chứa các điểm dừng *đủ sớm* với mọi đỉnh i, bất kì đường con nào cũng được biểu diễn bởi một đường đi trong $mangrove$ hoặc $mangrove$ hoàn chỉnh với UTT là cận dưới của thời gian di chuyển. Không mất tính tổng quát, thuật toán khởi tạo L^i với điểm dừng $(i,0)$ cho mỗi i. (Trong thực nghiệm, việc dùng (i,t) với t là thời điểm ngay sau s nếu (i,s) thuộc $\mathcal{F}^{1,0}$ sẽ hiệu quả hơn). Theo cách tương tự, tôi thêm (i,T) vào L^i và coi $(i,T) - mangrove$ được giải quyết và UTT cho các cung là thời gian di chuyển thực tế.

Các $i - mangrove$ cho các i khác nhau phải được kết nối bằng các cung chờ đợi tại một nút nằm giữa các đường con.

Nhắc lại: Thuật toán sẽ duy trì một TEN có kí hiệu $\mathcal{D}_{LB}$, với các UTT liên quan trên mỗi cung, và được định nghĩa là một hàm của các $mangrove$ và $mangrove$ hoàn chỉnh có các điểm dừng thuộc tập $\bigcup_{i\in N} L^i$ như sau:

Các nút và các cung: Tất cả các nút và cung trong $mangrove$ được giải quyết và $mangrove$ hoàn chỉnh đều được đưa vào $\mathcal{D}_{LB}$ cùng với các UTT của chúng.

Cung chờ: Các cung chờ được đưa vào để kết nối $mangrove$ (hoàn chỉnh)[1] đại diện cho một đường con đến một mangrove (hoàn chỉnh) khác có thể đại

[1]Mangrove hoặc mangrove hoàn chỉnh đã được giải quyết.

diện cho đường con kế tiếp. Có nghĩa là sẽ xem xét bất kỳ đường con i nào kết thúc tại đỉnh j, sau đó chờ tại j, kế tiếp là đường con k bắt đầu tại j (với $k \neq i$) trong khi đảm bảo UTT của nó không lớn hơn thời gian thực tế. Ta chỉ xét thời gian chờ dương tại j.

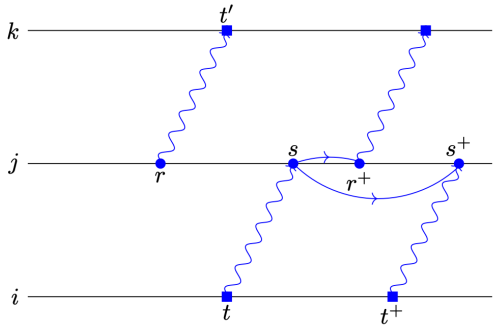


Figure 8: Forwards waiting arcs: Cases 1 and 2(a).

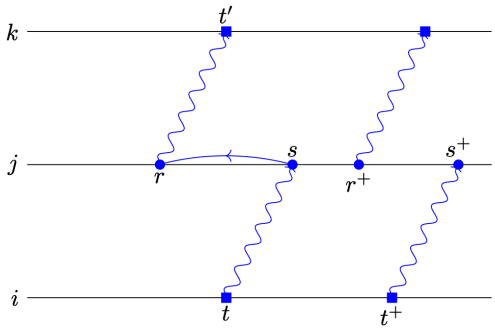


Figure 9: Backwards waiting arc: Case 2(b).

Hình 4.2: Hình vuông nhỏ kí hiệu cho "gốc" của $mangrove$, hình tròn biểu diễn cho một nút và đường con được biểu diễn bởi đường xoắn màu xanh biển.

Các trường hợp có thể xảy ra như sau:

TH1: Để biểu diễn cung chờ tại đỉnh j đến thời điểm s^+ hoặc muộn hơn, ta thêm cung $((j,s),(j,s^+))$ vào $\mathcal{D}_{LB}$ và gán UTT bằng 0. Các cung chờ nối các nút có trong các BSPT cho $mangrove - i$ liên tiếp thuộc L^i là không cần thiết nên ta có thể giả sử rằng việc chờ đợi chỉ xảy ra ở cuối một đường con chứ không phải ở đầu (không mất tính tổng quát).

TH2: Đối với mỗi đỉnh $k \neq i$, lấy r là thời điểm ngay trước s sao cho nút (j,r) xuất hiện trong BSPT cho một $mangrove - k$, giả sử là $\mathcal{M}^{k,t'}$. (Cách chọn này sẽ không tồn tại $t'' > t'$ với $\mathcal{M}^{k,t''}$ trong L^k có chứa (j,r') trong $\mathcal{B}^{k,t''}$ với $r < r' < s$.) Chắc chắn phải tồn tại t' và r như vậy, vì (j,s) trong $\mathcal{F}^{i,t}$ đồng nghĩa với việc hoặc $s > 0$, hoặc $i = j$ và $s = t = 0$, trong khi bất kì nút nào trong $\mathcal{B}^{k,0}$ đều có thời gian âm ngoại trừ ở k và tôi đã khởi tại L^k với $(k,0)$ nằm trong. Ngoài ra, nếu $t' < T$, lấy r^+ sao cho (j,r^+) nằm trong BSPT cho $mangrove$ có điểm dừng tiếp theo trong L^k sau (k,t'). Lưu ý $r^+ \geq s$. Xét hai trường hợp phụ sau:

a. $Mangrove$ cho (k,t') đã được giải quyết:
Không cần cung chờ nào nối (j,s) với $mangrove$ cho (k,t') vì $mangrove$ này chỉ biểu diễn các đường con (k,t') và không thể đến được nút (k,t') với thời gian chờ dương tại (j,s). - Nếu $t' = T$ thì không có $k-mangrove$ tiếp theo sau $(k,t')-mangrove$. -

Ngược lại, (j, s) được nối với $k-mangrove$ tiếp theo: thêm cung $((j, s), (j, r^+))$ vào $\mathcal{D}_{LB}$, với UTT bằng 0, điều này cho phép $\mathcal{D}_{LB}$ biểu diễn chuỗi của: đường con i kết thúc tại j ở thời điểm $r \in [s, s^+)$, đường con k bắt đầu tại j ở thời điểm $\tau \in [s, s^+)$, và đường con k bắt đầu tại j ở thời điểm $\tau' > \tau$, sử dụng điểm dừng tiếp theo tại k sau t'. Vì $r^+ \geq s$ nên gọi $((j, s), (j, r^+))$ là **cung chờ tiến.**

b. $Mangrove$ cho (k, t') chưa được giải quyết:
Cung $((j, s), (j, r))$ được đưa vào $\mathcal{D}_{LB}$ với UTT bằng 0. Lúc này, $\mathcal{D}_{LB}$ có thể biểu diễn chuỗi của một đường con i kết thúc tại j ở thời điểm $\tau \in [s, s+)$ và đường con k bắt đầu tại j ở thời điểm $\tau' \in [\tau, r^+)$. Vì $r \leq s$ nên gọi $((j, s), (j, r))$ là **cung chờ lùi.**

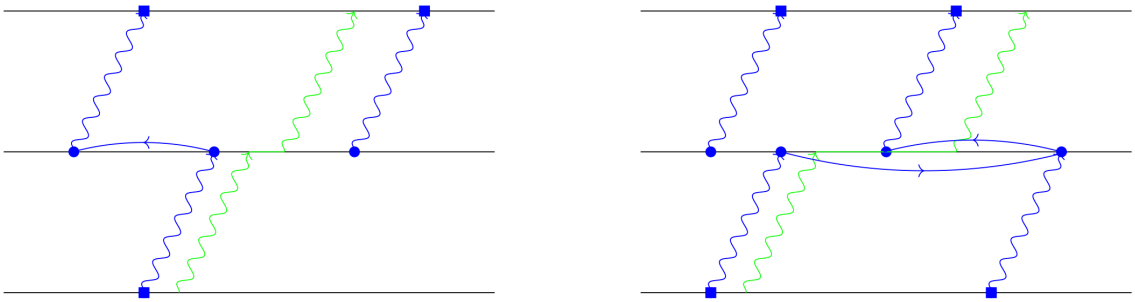


Figure 10: Lower bound paths using waiting arcs allow possible optimal paths to be represented.

Hình 4.3: Cách các cung chờ giúp TEN tìm được các đường đi tối ưu (màu xanh lá cây)

Các nút và cung ảo: Thêm một nút bắt đầu ảo và một nút kết thúc ảo vào $\mathcal{D}_{LB}$. Thêm các cung nối nút bắt đầu ảo với các nút có dạng $(1, t)$ với $t \geq 0$ và các nút có dạng (n, t) với $t \leq T$ nối với nút kết thúc ảo. Tất cả các cung thêm vào đều có UTT bằng 0.

Đường đi có UTT nhỏ nhất trong $\mathcal{D}_{LB}$ được xây dựng như trên, tính từ nút bắt đầu ảo đến nút kết thúc ảo, giá trị UTT sẽ là cận dưới cho bài toán MTTP. Điều này được giải thích dưới đây.

Xét một nghiệm tối ưu trong MTTP, gồm hai đoạn con:

- Đoạn con (i, α) P_1^*: Di chuyển từ nút (i, α) đến (j, α') mà không chờ đợi.
- Đoạn con (k, β') P_2^*: Di chuyển từ nút (j, β) đến (k, β'), với $\beta - \alpha' > 0$ là thời gian chờ đợi tại điểm j.

Gọi (i, s) là điểm cuối của danh sách L^i với $s \leq \alpha$ và (k, t') là điểm cuối của L^k với $t' \leq \beta'$. Lúc này P_1^* là đường đi trong $\overline{\mathcal{M}}^{i,s}$ với UTT cận trên là

thời gian di chuyển của P_1^* và tương tự với P_2^* cho $\overline{\mathcal{M}}^{k,t'}$. Từ $\overline{\mathcal{F}}^{i,s}$ cho ra duy nhất nút (j, s') và $\overline{\mathcal{B}}^{k,t'}$ cho ra duy nhất nút (j, t) đỉnh j. Luôn có một đường đi trong $\mathcal{D}_{LB}$ từ (j, s') đến (j, t) với UTT bằng 0, theo như cách xây dựng ở trên. Quan sát rằng $s' \leq \alpha$ và $t \leq \beta$ bởi thứ tự đã được sắp xếp của các $mangrove$ cho cùng đỉnh.

Tuy nhiên $t - s'$ có thể âm:

- Nếu $t < s'$ thì $(k, t') - mangrove$ không thể giải (vì nếu ngược lại, bắt buộc $t' = \beta'$ và $t = \beta > \alpha' \geq s'$), vậy nên cung chờ lùi $((j, t), (j, s'))$ được thêm vào $\mathcal{D}_{LB}$ như trong Trường hợp 2(b).
- Nếu $t \geq s'$, có 2 trường hợp con như sau:
  - $(k, t') - mangrove$ giải được, cung chờ tiến $((j, t), (j, s'))$ được thêm vào $\mathcal{D}_{LB}$ như trong Trường hợp 2(a),
  - Ngược lại, gọi (i, r) là điểm đầu tiên ngay sau (i, s) trong danh sách L^i sao cho điểm duy nhất (j, r') từ $\mathcal{F}^{i,r}$ có $r' \geq t$. (r luôn tồn tại vì $j \neq n$ và (i, T) thuộc L^i). Lúc này:
    - Theo Trường hợp 1, sẽ có một chuỗi các cung có UTT$= 0$ nối (j, s') với (j, r'), và
    - Theo trường hợp 2(b), sẽ có cung chờ lùi $((j, r'), (j, s'))$ có UTT $= 0$ trong $\mathcal{D}_{LB}$.

Kết luận: Trong mọi trường hợp, luôn có một đường đi UTT bằng 0 trong (j, t) đi từ (j, s') đến (j, t). Do đó, đoạn con P_1^* nối tiếp P_2^* được biểu diễn trong $\mathcal{D}_{LB}$ với một đường đi có UTT không lớn hơn tổng thời gian di chuyển của P_1^* và P_2^*.

4.3 Mô tả thuật toán

Proposition 3. *Sử dụng quy trình ở trên, chúng ta có thể xây dựng $\mathcal{D}_{LB}$ và UTT cho mỗi cung trong $\mathcal{D}_{LB}$ từ các danh sách L^i có các nút $(i, 0), (i, T) \in L^i$ với mỗi đỉnh $i \in N$. Sau đó, đường đi UTT ngắn nhất trong $\mathcal{D}_{LB}$ từ nút bắt đầu ảo đến nút kết thúc ảo có UTT là **cận dưới** của MTTP.*

Thuật toán sử dụng hai hàm như sau:

- $createLBTEN(\{L^i\}_{i \in N}, Resolved)$: Xây dựng $\mathcal{D}_{LB}$ và các UTT tương ứng, được lưu lại trong vector $\underline{c}$ (như đã mô tả ở trên). $Resolved$ là tập hợp các điểm mốc đã được giải quyết.

- $computeSP(\mathcal{D}_{LB}, c)$: Tính toán đường đi có UTT nhỏ nhất trong $\mathcal{D}_{LB}$, trả về cả đường đi và tổng UTT của nó.

Thêm vào đó, các danh sách $\{L^i\}_{i \in N}$ cũng được dùng để tạo ra **cận trên** bằng cách xây dựng một mạng TEN kí hiệu là $\mathcal{D}_{UB}$ theo các bước như sau:

1. Tất cả các cung thuộc $(i,t) - mangrove$ với (i,t) thuộc L^i và tất cả $i \in N$ đều cho vào $\mathcal{D}_{UB}$, với thời gian di chuyển ban đầu.
2. Với mỗi (j,s) trong FSPT của một $i - mangrove$ nào đó và với mỗi $k \neq i$, tìm r là thời điểm ngay sau s mà nút (j,r) xuất hiện trong BSPT của một $k - mangrove$ (nếu có). Nếu tìm được r thoả mãn, thêm cung đợi $((j,s),(j,r))$ vào $\mathcal{D}_{UB}$ và đặt thời gian di chuyển là 0.
3. Thêm một nút bắt đầu ảo và kết thúc ảo vào $\mathcal{D}_{UB}$. Thêm các cung để nối nút bắt đầu ảo với các nút có dạng $(1,t)$ với $t \geq 0$, tương tự với các nút có dạng (n,t) với $t \leq T$ nối đến nút kết thúc ảo. Tất cả cung được thêm vào đều có thời gian di chuyển là 0.

Bất kì đường đi nào trong mạng $\mathcal{D}_{UB}$, từ nút bắt đầu ảo đến kết thúc ảo, đều là một nghiệm khả thi, và tổng thời gian di chuyển tạo ra **cận trên** cho bài toán MTTP. Bằng cách tối thiểu hoá thời gian di chuyển, ta sẽ dần tìm ra cận trên tốt hơn từ $\mathcal{D}_{UB}$.

Thuật toán sẽ sử dụng hai hàm dưới đây để tạo ra mạng $\mathcal{D}_{UB}$:

- $createUBTEN(\{L^i\}_{i \in N})$: Xây dựng mạng $\mathcal{D}_{UB}$, và lưu lại các thời gian di chuyển tương ứng trong vector $\overline{c}$ (như đã mô tả ở trên).
- $computeSP(\mathcal{D}_{UB}, \overline{c})$: Tìm đường đi khả thi tốt nhất ở hiện tại cho MTTP, trả về cả đường đi và tổng thời gian di chuyển của nó.

Sau khi có các công cụ cần thiết, ta tiến hành xây dựng thuật toán 2 cho bài toán MTTP theo hai bước như sau:

B1. Khởi tạo:

- Xây dựng hai mạng:
  - Mạng TEN $\mathcal{D}_{LB}$,
  - Mạng TEN $\mathcal{D}_{UB}$.
- Tìm cận trên và cận dưới cho thời gian di chuyển bằng cách tìm đường đi trên mỗi mạng.

B2. Lặp lại bước này cho đến khi 2 cận bằng nhau:

- Nếu hai cận chưa bằng nhau:

  - Xét đường đi có UTT nhỏ nhất trong $\mathcal{D}_{LB}$ và $mangrove$ hoàn chỉnh đã được dùng.
  - Ít nhất một trong các $(i,t) - mangrove$ đã được dùng có điểm dừng (i,t) chưa được giải quyết, nếu không thì cận trên và cận dưới đã bằng nhau.
  - Thuật toán chọn ra một tập bao gồm một hay nhiều điểm dừng chưa giải quyết như vậy.
  - Đối với mỗi điểm dừng (giả sử là (i,t)) trong tập đã chọn:
    * Chọn một số điểm dừng (một hoặc nhiều) giữa (i,t) và điểm tiếp theo trong L^i,
    * Thêm các điểm dừng đã chọn vào L^i,
    * Nếu có bất kì nút nào trong L^i ngay sau điểm dừng (i,s) cũng thuộc L^i, ta coi (i,s) đã được giải quyết.

- Tiếp tục tạo các $\mathcal{D}_{LB}$ và $\mathcal{D}_{UB}$ dựa trên L^i đã cập nhật, sau đó tính toán lại các cận trên và dưới.

Hiệu năng của thuật toán này mấu chốt nằm ở hai vấn đề sau:

1. **Tập điểm dừng chưa giải quyết:** Trong mỗi lần lặp, thuật toán cần chọn một tập các điểm dừng chưa được giải quyết,
2. **Tập điểm dừng cho cùng đỉnh:** Trong mỗi lần lặp, cần chọn tập điểm dừng có chung một đỉnh để đưa vào danh sách của đỉnh đó.

Chi tiết về vấn đề này sẽ được đề cập ở **??**. Ở đây sẽ không đi vào chi tiết (để đơn giản hoá cho mã giả). Hàm $FindBP(P, \{L^i\}_{i \in N})$ được dùng để xác định tập các BP mới thêm vào danh sách L^i, dựa trên đường đi có UTT nhỏ nhất P tìm được từ $\mathcal{D}_{LB}$.

4.4 Mã giả

Algorithm 2 Dynamic Discretization Discovery Algorithm for the MTTP

Input: digraph $D = (N, A)$, latest time at end node T, arc travel time function $c_{i,j}(t)$ for all $t \in [0, T]$, for each $(i, j) \in A$

Output: minimum travel time path from node 1 to n departing from 1 and arriving at n at times in $[0, T]$

$Resolved := \emptyset$;
for $i \leftarrow 1$ **to** n **do**
  Compute the $(i, 0)$-mangrove and the arc-completed $(i, 0)$-mangrove;
  Compute the (i, T)-mangrove and the arc-completed (i, T)-mangrove;
  Set $L^i := [(i, 0), (i, T)]$;
  $Resolved := Resolved \cup (i, T)$;
end for
Initialize $LB = -\infty$ and $UB = \infty$;
while $(LB < UB)$ **do**
  Construct lower and upper bound TENs, with their UTTs and travel times, respectively:
  $(\mathcal{D}_{LB}, \underline{c}) \leftarrow createLBTEN(\{L^i\}_{i \in N}, Resolved)$,
  $(\mathcal{D}_{UB}, \overline{c}) \leftarrow createUBTEN(\{L^i\}_{i \in N})$;
  Compute current lower and upper bounds along with their corresponding path:
  $(P_{LB}, LB) \leftarrow computeSP(D_{LB}, \underline{c}), (P_{UB}, UB) \leftarrow computeSP(D_{UB}, \overline{c})$;
  $BP \leftarrow findBP(P_{LB}, \{L^i\}_{i \in N})$;
  for $(j, t) \in BP$ **do**
  end for
  Compute the (j, t)-mangrove and the arc-completed (j, t)-mangrove;
  Insert (j, t) in the list L^j;
  if (j, t^-) is in L^j immediately before (j, t) and no breakpoint at j is between them **then**
    $Resolved := Resolved \cup (j, t-)$
  end if
  if (j, t^+) is in L^j immediately after (j, t) and no breakpoint at j is between them **then**
    $Resolved := Resolved \cup (j, t)$
  end if
end while
return (UB, P_{UB})

5 Các thử nghiệm

Để tiến hành đánh giá hiệu năng thuật toán của các chương trước, tôi sử dụng dữ liệu được sinh ngẫu nhiên để chạy. Thuật toán được cài đặt và chạy bằng ngôn ngữ C++17, trên máy tính để bản với hệ điều hành Fedora 39, CPU Intel i7-8700 (12) @ 4.600GHz và 128GB RAM.

5.1 Tập dữ liệu

Dữ liệu được chuẩn bị cho việc chạy các thuật toán. Có 4 kiểu dữ liệu tương ứng với 4 kiểu mạng $D = (N, A)$ và tập các đỉnh $N = \{1, 2, \dots, n\}$;

- **Kiểu 1:** $A = \{(i, j) \in N \times N | i < j\}$;
- **Kiểu 2:** $A = \{(i, i+1) | i = 1, \dots, n-1\} \cup \{(i, j) \in N \times N | i < j+1 \text{ và } U_{i,j} < 0.5\}$ với mỗi (i, j), giá trị $U_{i,j}$ được lấy độc lập từ phân phối chuẩn $[0, 1]$ hay $U_{i,j} \sim U[0, 1]$
- **Kiểu 3:** $A = \{(i, j) \in N \times N | i < j < i + 4\}$;
- **Kiểu 4:** $A = \{(i, j+1) | i = 1, \dots, n-1\} \cup \{(i, j) \in N \times N | i < j + 1 \text{ và } U_{i,j} < 1/|j - i|\}$ với $U_{i,j} \sim U[0, 1]$ cho mỗi (i, j).

5.1.1 Tóm tắt cho mỗi kiểu dữ liệu

Kiểu 1:

- Mỗi cặp đỉnh bất kì đều được nối với nhau bằng một cung.
- Mật độ của mạng: 0.5 (mỗi đỉnh kết nối đến tất cả các đỉnh khác).

Kiểu 2:

- Bao gồm nửa số cung có trong Kiểu 1,
- Để đảm bảo luôn có ít nhất một đường đi khả thi, mọi cung giữa các đỉnh liên tiếp vẫn được giữ lại.
- Mật độ mạng: khoảng 0.25.

Kiểu 3:

- Mỗi đỉnh được nối với ba đỉnh tiếp theo theo số thứ tự.
  - Ví dụ: đỉnh 1 sẽ nối với đỉnh 2, 3 và 4.
- Mật độ mạng: khoảng $3/n$.

Kiểu 4:

- Khả năng tồn tại một cung giữa hai đỉnh phụ thuộc vào khoảng cách giữa chúng.
- Càng xa nhau về mặt số thứ tự thì càng giảm khả năng xuất hiện cung.
- Để đảm bảo luôn có ít nhất một đường đi khả thi, tất cả các cung giữa các đỉnh liên tiếp vẫn được giữ lại.
- Mật độ mạng: khoảng $\log n!/(n^2)$.

5.2 Hàm thời gian

Để đánh giá mức độ ảnh hưởng của hàm thời gian di chuyển đến hiệu năng của thuật toán, có ba loại hàm được sử dụng. **??** minh hoạ cho mỗi loại. Hàm thời gian di chuyển được tạo ra như sau: với mỗi cung (i, j), sử dụng nội suy tuyến tính từng khúc của một hàm f (mô tả chi tiết ở dưới) với các điểm là số nguyên.

Loại 1: f là đa thức bậc 4:

$$f([0, \frac{T}{4}, \frac{T}{2}, \frac{3T}{4}, T]) = \begin{cases} [1.6, 1, 1.05, 1, 1.6](B_{i,j}\frac{|j-i|}{10}), & \text{nếu } U_{i,j} < \frac{1}{3}, \\ [2, 1, 1.5, 1, 2](B_{i,j}\frac{|j-i|}{10}), & \text{nếu } \frac{1}{3} \le U_{i,j} < \frac{2}{3}, \\ [2.5, 1, 1.75, 1, 2.5](B_{i,j}\frac{|j-i|}{10}), & \text{còn lại,} \end{cases} \quad (5.1)$$

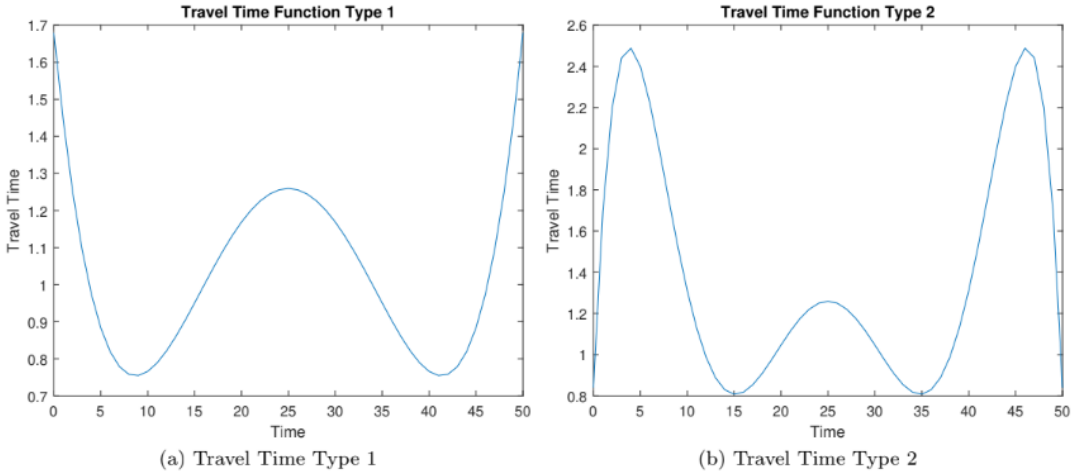
Với $B_{i,j}, U_{i,j} \sim U[0, 1]$.

Loại 2: f là đa thức bậc 6:

$$f([0, \frac{T}{6}, \frac{T}{3}, \frac{T}{2}, \frac{2T}{3}, \frac{5T}{6}, T]) = \begin{cases} [1, 1.6, 1, 1.05, 1, 1.6, 1](B_{i,j}\frac{|j-i|}{10}), & \text{nếu } U_{i,j} < \frac{1}{3}, \\ [1, 2, 1, 1.5, 1, 2, 1](B_{i,j}\frac{|j-i|}{10}), & \text{nếu } \frac{1}{3} \le U_{i,j} < \frac{2}{3}, \\ [1, 2.5, 1, 1.75, 1, 2.5, 1](B_{i,j}\frac{|j-i|}{10}), & \text{còn lại,} \end{cases} \quad (5.2)$$

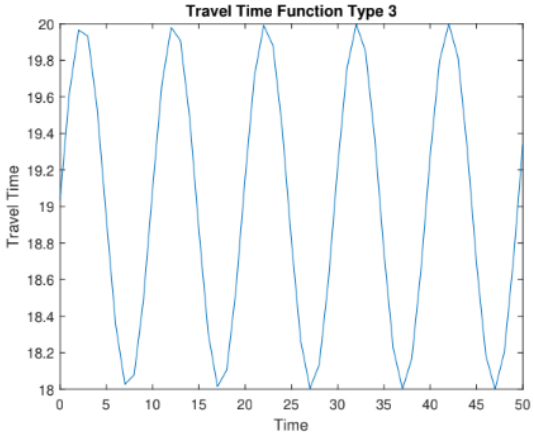
Với $B_{i,j}, U_{i,j} \sim U[0,1]$.

Loại 3: $f(t) = (j-i) + sin(B_{i,j} \times t)$, với $B_{i,j} \sim U[0,1]$.



(a) Travel Time Type 1

(b) Travel Time Type 2



(c) Travel Time Type 3

Figure 11: Plots of different travel time function types

Hình 5.1: Figure 11

5.2.1 Nguồn gốc của các hàm thời gian

Hàm **Loại 1** được lấy ý tưởng từ [6] về việc chia 5 khoảng thời gian bằng nhau, mỗi khoảng có tốc độ di chuyển không đổi. Tốc độ di chuyển cơ sở (lấy ngẫu nhiên) được nhân với các hệ số lần lượt là $1.6, 1.0, 1.05, 1.0$ và 1.6 tại các thời điểm 0, $\frac{1}{4}T$, $\frac{1}{2}T$, $\frac{3}{4}T$ và T tương ứng. Sau đó sử dụng nội suy đa thức qua các điểm này để tạo nên hàm liên tục. Cuối cùng chuyển chúng về dạng hàm tuyến tính từng khúc bằng cách nội suy tuyến tính từng khúc của đa thức trên tại các điểm nguyên.

Hàm **Loại 2** được mở rộng từ **Loại 1** bằng cách thêm hai điểm ở đầu và cuối để tăng độ khó (tăng bậc).

Hàm **Loại 3** gây ra nhiều thách thức cho thuật toán vì chúng không đồng pha trên các cung có nhiều điểm cực tiểu và cực đại cục bộ.

Ngoài ra các hàm khác được đề xuất trong [6] cũng được thử nghiệm, nhưng thuật toán này tìm ra nghiệm chỉ sau vài lần lặp. Các nghiệm tối ưu thường bắt đầu ở thời điểm sớm nhất hoặc muộn nhất có thể, do đó không giúp ích cho việc phân tích hiệu năng của thuật toán.

Hiệu năng của cả hai bài toán với phương pháp DDD phụ thuộc rất nhiều vào việc tính toán UTT: có thể tính toán một cách hiệu quả thời gian tối thiểu di chuyển qua các cung ngay khi bắt đầu chạy thuật toán. Bảng tra cứu là một cấu trúc dữ liệu hiệu quả cho việc tính toán trước khi chạy.

5.3 Kịch bản và cài đặt

Phần này sẽ bàn về ảnh hưởng của việc chọn điểm dừng đến hiệu năng của thuật toán.

Với bài toán MDP, giả sử cận dưới đang là LB^t và t^+ là thời gian đến đỉnh n của ABSPT ngay sau $\overline{\mathcal{B}}^t$. Kí hiệu cho nút trong $\overline{\mathcal{B}}^t$ là (i, t_i) và trong $\overline{\mathcal{B}}^{t^+}$ là (i, t_i^+). Chọn cung (i, j) sao cho mỗi cung thuộc $\overline{\mathcal{B}}^t$ có điểm dừng trong khoảng thời gian (t_i, t_i^+) và có chỉ số nhỏ nhất (khi xét các đỉnh từ 1 đến n). Dưới đây là ba cách chọn điểm gãy cho $c_{i,j}(t)$ với $t \in (t_i, t_i^+)$:

1. Chọn một điểm bất kì ($RAND$)
2. Chọn điểm trung vị (trong tập các điểm dừng sắp xếp theo thời gian) (MED)
3. Chọn điểm có giá trị $c_{i,j}(t)$ nhỏ nhất với $t \in (t_i, t_i^+)$ (MIN).

Với bài toán MTTP, ta so sánh hai kịch bản chọn điểm dừng để thêm vào danh sách L^i ở mỗi lần lặp như sau:

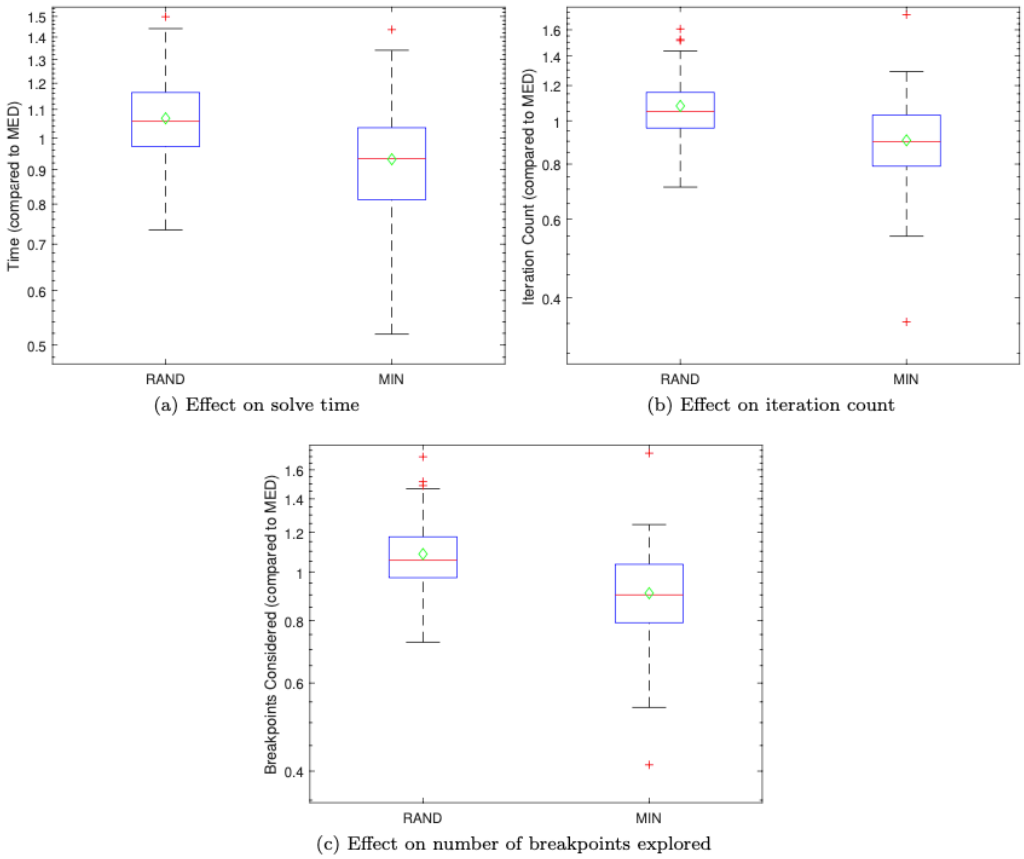
- **Thêm một điểm dừng (S):** Thêm một điểm dừng cho một trong các đường con ở mỗi lần lặp.
- **Thêm nhiều điểm dừng (M):** Thêm một điểm dừng cho mọi đường con ở mỗi lần lặp.

Quy tắc chọn điểm dừng như sau: Để chọn điểm cần thêm vào, ta xác định một chuỗi các $mangrove$ hoàn chỉnh được sử dụng bởi đường đi UTT kí hiệu P_{LB}. Mỗi $mangrove$ hoàn chỉnh được tạo ra từ một BP, giả sử đó là điểm (i, t). Khi đó, nếu (i, t^+) là điểm ngay sau (i, t) trong L^i và có một BP chưa được giải quyết tại đỉnh i có thời gian giữa t và t^+, thì điểm (i, t^+) này sẽ xem

xét để chọn. Tập hợp các điểm như vậy gọi là tập ứng viên. Cách chọn điểm từ tập ứng viên thuộc một trong ba cách ở trên ($RAND, MED, MIN$). Với kịch bản **(S)**, thêm một BP của đỉnh i cho $i - mangrove$ hoàn chỉnh đầu tiên được dùng bởi đường đi UTT từ 1 đến n, nếu BP ứng viên tồn tại. Trong kịch bản **(M)**, thêm một BP vào mỗi $mangrove$ hoàn chỉnh được dùng bởi đường đi UTT, nếu BP ứng viên tương ứng tồn tại.

Đánh giá hiệu năng của việc lựa chọn BP:

Đầu tiên, với bài toán MDP, kết quả được biểu diễn ở **??** bằng biểu đồ hộp với các tỉ số: thời gian chạy, số lần lặp và số BP được giải quyết cho cách chọn $RAND$ và MIN khi so sánh với MED.



(a) Effect on solve time (b) Effect on iteration count

(c) Effect on number of breakpoints explored

Figure 12: Comparison of breakpoint selection schemes on 120 instances of the MDP with $n = 20$, $T = 50$

Hình 5.2: So sánh các cách chọn với 120 mẫu của bài toán MDP cho $n = 20, T = 50$

- **Biểu đồ hộp:**
  - Hình thoi màu xanh lá cây biểu thị giá trị trung bình.
  - Đường ngang màu đỏ biểu thị giá trị trung vị.
- **Trục Y:** thang đo logarit.