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 đại học hệ chính quy
Ngành: Khoa học máy tính và thông tin
(Chương trình đào tạo chuẩn)

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 đại học hệ chính quy
Ngành: Khoa học máy tính và thông tin
(Chương trình đào tạo chuẩn)

Cán bộ hướng dẫn: TS. Vũ Đức Minh

Hà Nội - 2024

Lời cảm ơn

Trong những năm tháng sinh viên, em đã được học hỏi và trải qua nhiều điều, trong đó có những khó khăn và thách thức. Em xin bày tỏ lòng biết ơn sâu sắc đến bạn bè, thầy cô và gia đình đã luôn ở bên cạnh, hỗ trợ và giúp đỡ em trong suốt thời gian qua. Các thầy cô ở khoa Toán - Cơ - Tin học, trường Đại học Khoa học Tự nhiên, ĐHQG Hà Nội đã luôn tạo điều kiện tốt nhất trong quá trình học tập và phát triển.

Khóa luận "**Phương pháp rời rạc động giải các bài toán tìm đường đi có yếu tố thời gian**"của em đã không thể hoàn thành mà không có sự hướng dẫn, giúp đỡ tận tình của thầy **TS. Vũ Đức Minh**. Em xin chân thành cảm ơn thầy đã dành thời gian để chỉ bảo em tận tình trong quá trình thực hiện khóa luận này. Cuối cùng, em xin chân thành cảm ơn tất cả mọi người đã giúp đỡ em trong suốt thời gian qua.

Mặc dù đã có nhiều cố gắng hoàn thiện nhưng do hạn chế ở kĩ năng và kinh nghiệm của bản thân nên sẽ không tránh khỏi những thiếu sót còn tồn đọng. Em rất mong nhận được sự góp ý, chỉ bảo từ quý thầy cô và các bạn để khóa luận của em trở nên hoàn thiện hơn.

Em xin chân thành cảm ơn!

Hà Nội, ngày 19 tháng 5 năm 2024

Bùi Khánh Duy

Mở đầu

Tìm đường đi ngắn nhất trong mạng vốn là một trong những bài toán tối ưu cơ bản. Tuy nhiên những biến thể của nó vẫn luôn hấp dẫn và có ý nghĩa thực tiễn cao. Các ứng dụng của bài toán này rất đa dạng, từ việc tìm đường đi cho mô hình mạng lưới giao thông đến các bài toán vận tải. Ngoài ra, đường đi ngắn nhất không dừng lại ở khoảng cách, nó còn có thể là nhỏ nhất về thời gian, chi phí, hoặc một tiêu chí nào đó khác.

Khóa luận sẽ tập trung vào bài toán thời gian di chuyển nhỏ nhất, với việc thời gian di chuyển trên một cung (trọng số của cung) trong mạng phụ thuộc vào **thời gian khởi hành** di chuyển trên cung đó. Điều này có ý nghĩa thực tiễn, khi không phải lúc nào việc đến đích càng sớm càng tốt cũng mục tiêu duy nhất. Các mục tiêu khác nhau như giảm thiểu sự chênh lệch giữa thời gian đến đích và thời gian khởi hành, hay giảm thiểu tổng thời gian di chuyển theo đường đi từ nguồn đến đích đều là những đáng để xem xét. Phương pháp được giới thiệu là thuật toán khám phá rời rạc động (DDD), với thời gian di chuyển là các hàm vận tốc phụ thuộc vào thời gian, tuyến tính từng khúc. Thuật toán hoạt động trên các mạng thời gian (TEN) trong đó chi phí của cung sẽ đại diện bởi các cận dưới về thời gian di chuyển trong khoảng thời gian tiếp theo. Một đường đi ngắn nhất trong mạng TEN này tạo ra cận dưới và cận trên cho giá trị của nghiệm tối ưu. Thuật toán liên tục tinh chỉnh sự rời rạc bằng cách khai thác các thời điểm dừng (BP) của các hàm vận tốc, loại bỏ các khoảng thời gian thừa, cải thiện cận dưới và cận trên cho đến khi, trong một số lần lặp hữu hạn, cả hai cận hội tụ (thu được nghiệm tối ưu). Các thử nghiệm cho thấy chỉ có một phần nhỏ các BP cần được khám phá, và tỷ lệ này giảm đi khi độ dài của khung thời gian và kích thước của mạng tăng lên, làm cho các thuật toán vô cùng hiệu quả và có khả năng mở rộng cao.

Bố cục của khóa luận như sau:

- Chương : Giới thiệu.
  Trình bày một số khái niệm, phát biểu bài toán và các công trình liên quan.
- Chương 1: Bài toán MDP.
  Trình bày bài toán tìm đường đi có thời gian di chuyển nhỏ nhất phụ thuộc vào hàm vận tốc (MDP), mô hình hóa và giải bài toán thông qua mã giả và ví dụ minh họa.
- Chương 2: Các thử nghiệm.
  Trình bày các thử nghiệm và kết quả chạy của thuật toán DDD và ví dụ thực tế.
- Chương : Kết luận.
  Tóm tắt kết quả và đề xuất hướng phát triển tiếp theo.
- Chương : Phụ lục.
  Bao gồm các thông tin bổ sung, các hàm vận tốc cho ví dụ ở Chương 1 và toàn bộ kết quả chạy.

Từ khoá: Dynamic Discretization Discovery, Time-dependent Shortest Path Problem, Minimum Duration Time-Dependent Path Problem.

Danh mục viết tắt

Kí hiệu	Tên đầy đủ	Ý nghĩa
ABSPT	Arc-completed Backwards Shortest Path Tree	Cây đường đi ngắn nhất hoàn chỉnh lùi
BP	Breakpoint	Thời điểm dừng: thời điểm t làm thay đổi độ dốc của hàm vận tốc
BSPT	Backwards Shortest Path Tree	Cây đường đi ngắn nhất lùi
DDD	Dynamic Discretization Discovery	Khám phá rời rạc động
FIFO	First in first out	Vào trước ra trước
FSPT	Forwards Shortest Path Tree	Cây đường đi ngắn nhất tiến
MATP	Minimum Arrival Time Path Problem	Bài toán Đường đi thời gian đến tối thiểu
MDP	Minimum Duration Time-Dependent Path Problem	Bài toán Đường đi thời gian di chuyển tối thiểu
TDSP	Time-dependent shortest path	Đường đi ngắn nhất phụ thuộc thời gian
TDSPP	Time-Dependent Shortest Path Problem	Bài toán tìm đường đi ngắn nhất phụ thuộc thời gian
TEN	Time-expanded network	Mạng thời gian
UTT	Underestimated travel time	Cận dưới của thời gian di chuyển

Mục lục

Danh sách hình vẽ

Danh sách bảng

Danh sách thuật toán

Giới thiệu

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

Bên cạnh đó là 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 trong hai bài báo [1, 7] và được mô tả bởi bài [3] của tác giả Cooke và các cộng sự. Tổng quan về các phương pháp giải khác cho biến thể này được đề cập trong [4] với lý thuyết về sử dụng mạng FIFO.

Bài toán tìm đường đi thời gian tối thiểu (MDP) được nghiên cứu trong bối cảnh mạng lưới giao thông [5], thậm chí trong phân tích mạng xã hội [9]. Thuộc họ các bài toán Tìm đường đi ngắn nhất phụ thuộc thời gian (TDSPP) [10], trong đó thời gian di chuyển phụ thuộc thời gian thường là kết quả của sự tắc nghẽn. Thời gian di chuyển trên các cung thường được giả định thỏa mãn tính chất FIFO: không thể đi đến cuối của cung vào thời điểm sớm hơn so với thời điểm bắt đầu di chuyển từ đầu cung.

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. Việc xuất phát sớm hay muộn hơn có thể đi đến đích cùng lúc. Vậy nên việc lùi thời gian khởi hành có thể giảm tổng thời gian di chuyển.

Trong [2] có giới thiệu thuật toán khám phá rời rạc động (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

(TEN). Các mạng này được thiết kế để luôn tồn tại nghiệm là đường đi, tìm 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 xác định), mô hình quy hoạch nguyên sẽ cho ra kết quả tối ưu.

Phần mô tả bài toán, phương pháp giải quyết và kết quả từ khu vực Atlanta của khoá luận tham khảo từ bài báo [11]. Để phục vụ cho các phần sau, dưới đây là một số định nghĩa được sử dụng:

- **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 luôn là đường đi ngắn nhất.
- **FIFO:** Được lấy từ mạng FIFO trong [4]: đường đi từ s đến v là FIFO nếu với mọi cạnh (u, v) trên đường đi, thời gian bắt đầu di chuyển từ u đến v không nhỏ hơn thời gian bắt đầu di chuyển từ s đến u.

Mô tả bài toán

Cho một đồ thị có hướng

$$D = (N, A)$$

trong đó: $N = 1, 2, ..., n$ là tập đỉnh, $A \subseteq N \times N$ là tập cạnh; khung thời gian $[0, T]$ và các hàm vận tốc 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, nghĩa là độ dốc của các khúc lớn hơn hoặc bằng -1. Đồng thời việc chờ đợi ở bất kì đỉnh nào khác không phải nguồn được cho là không tối ưu.

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 $m(P)$ nút (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[1] 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 nút cần thỏa 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 $k \geq 2$ bất kì (hoặc $t_1^P > 0$ nếu $k = 1$), 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.

Hàm mục tiêu chính được quan tâm ở đây là **Tìm một đường đi có thời gian thực hiện tối thiểu (từ nguồn đến đích):**

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

Bên cạnh đó, ở một số trường hợp, ta cũng quan tâm đến mục tiêu phụ **Tìm một đường đi có thời gian đến đích tối thiểu:**

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

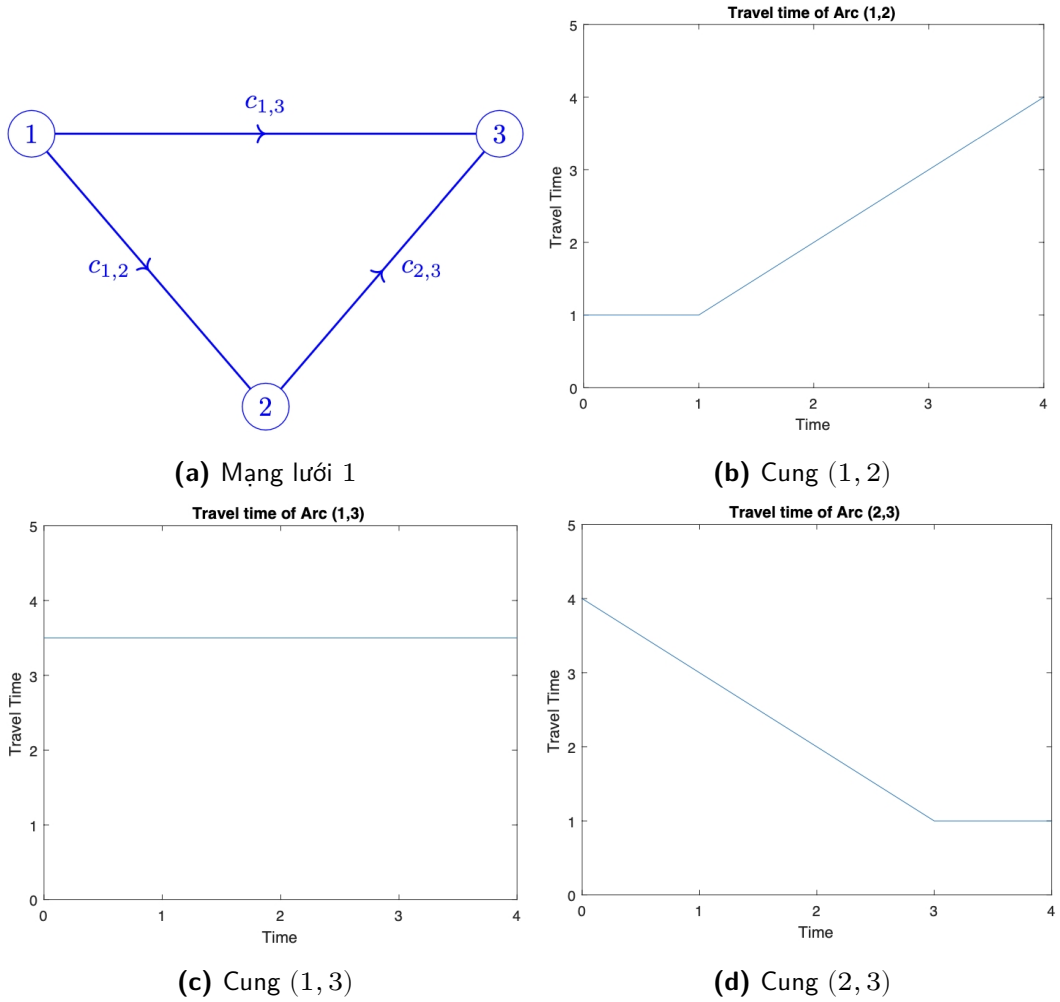
Trong phần còn lại của khóa luận sẽ bỏ đi phần chữ P ở trên các đỉnh cùng một đường đi để thuận tiện cho ký hiệu và dễ nhìn hơn.

Để minh họa hai hàm mục tiêu này, hãy xem xét đồ thị Hình 1a và các hàm vận tốc Hình 1(b,c,d), với $n = 3$ và khung thời gian 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.

[1] Tuy đỉnh và nút có nghĩa tương đồng, nhưng ở đây sẽ quy ước *đỉnh* là một số, *nút* là cặp (đỉnh, thời gian)

- **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.



(a) Mạng lưới 1 **(b)** Cung $(1, 2)$

(c) Cung $(1, 3)$ **(d)** Cung $(2, 3)$

Hình 1: Minh họa các hàm vận tốc

Vì các hàm vận tốc 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, 4, 12] 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. Điều 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 vận tốc *đảo chiều*:

Định nghĩa 1. *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 $t - \tau$.*

Từ ý tưởng của Định nghĩa 1 dẫn đến việc giải bài toán TDSPP với nghiệm là đường đi ngắn nhất TDSP ở các phần sau.

1 Bài toán MDP

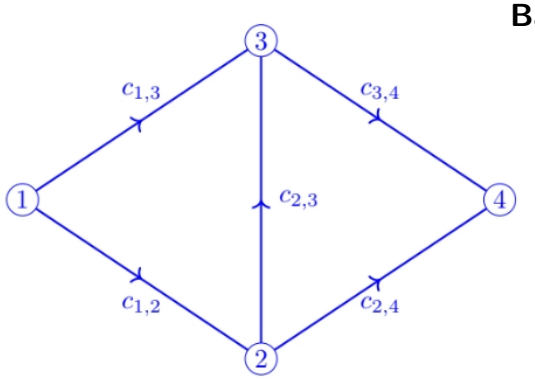
Khoá luận sẽ chỉ tập trung vào trường hợp mạng có thuộc tính *FIFO ngặt*: với mọi cung trong mạng, không thể đi đến cuối một cung sớm hơn bằng cách khởi hành ở thời điểm sau đó.

Trước khi đi vào các chi tiết về cách giải và ví dụ minh họa, cần làm rõ hai yếu tố sau:

- **Đầu vào:** Một mạng thời gian $D = (N, A)$; khung thời gian $[0, T]$; hàm vận tốc $c_{i,j}(t)$ tuyến tính từng khúc dưới dạng danh sách cạnh và tập hợp các giá trị tại từng BP. Các BP ở đây đều có thời gian là số nguyên.
- **Đầu ra:** Đường đi có thời gian di chuyển tối thiểu từ đỉnh nguồn đến đỉnh đích. Ngoài ra còn có các thông tin khác đề cập ở phần Chương 2.

1.1 Ví dụ

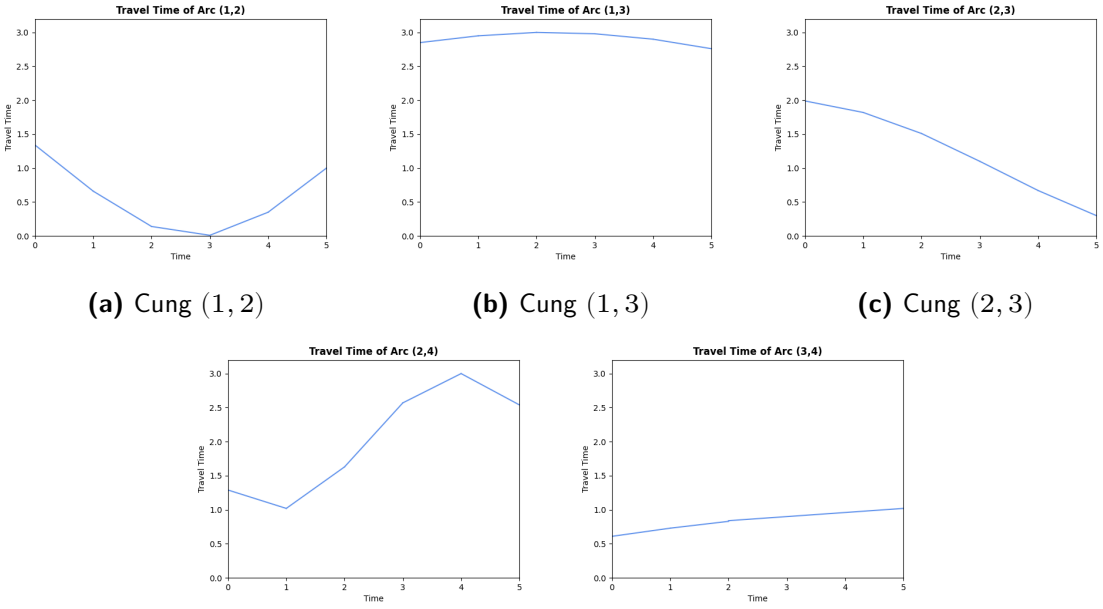
Xét mạng ở Hình 2 với các hàm vận tốc được biểu diễn ở Hình 3. Khung thời gian là $[0, 5]$. Hầu hết các cung có số BP là số nguyên. Ngoại lệ là cung $(3, 4)$ chỉ có BP tại $0, 1, 2$ và 5. Chương 2.7 cung cấp chi tiết về các hàm vận tốc và các hàm *đảo chiều* của ví dụ này.



Hình 2: Mạng D

Bảng 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
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



(a) Cung $(1, 2)$ **(b)** Cung $(1, 3)$ **(c)** Cung $(2, 3)$

(d) Cung $(2, 4)$ **(e)** Cung $(3, 4)$

Hình 3: Biểu đồ đường cho Hàm vận tốc.

1.2 Công thức chi tiết

Cho thời điểm t bất kỳ 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, luôn 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:

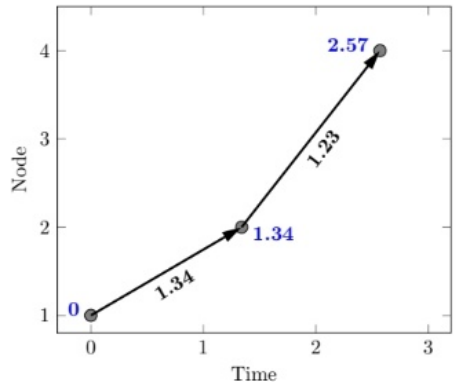
- Với mỗi $i \in N$, chỉ tồn tại t_i là thời gian khởi hành muộn nhất để đi từ i đến n. Lúc này $(i, t_i) \in B^t$,
- Cung $(i, j) \in A$,
- $t_i + c_{i,j}(t_i) = t_j$, và
- **Chỉ có một đường đi duy nhất** từ i đến n trong B^t. Đường đi này được xác định bởi việc thực hiện *TDSP* bắt đầu từ i tại thời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

BSPT được tạo từ nghiệm của TDSP (là một đường đi) bằng cách áp dụng thuật toán tìm đường đi ngắn nhất từ một đỉnh (Dijkstra) với một vài điều chỉnh đơn giản liên quan đến chọn các BP thay vì trọng số cố định như thông thường. BSPT cho đỉnh 4 tại thời gian $t = 2.57$ được biểu diễn ở Hình 4b. 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.

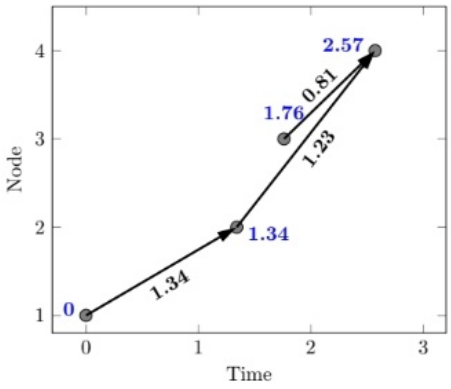
[1] để phân biệt mạng với đồ thị, sử dụng từ *nút* thay cho *đỉnh*.

[2] để phân biệt mạng với đồ thị, sử dụng từ *cung* thay cho *cạnh*.

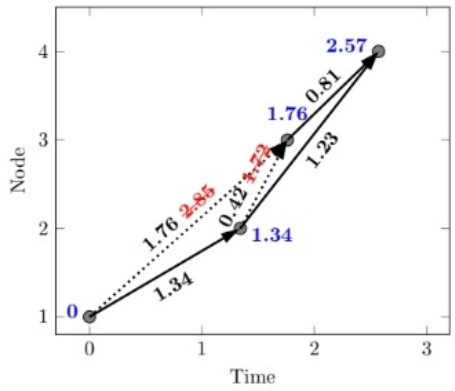
[3] thời gian làm tròn thời gian đến 2 chữ số thập phân



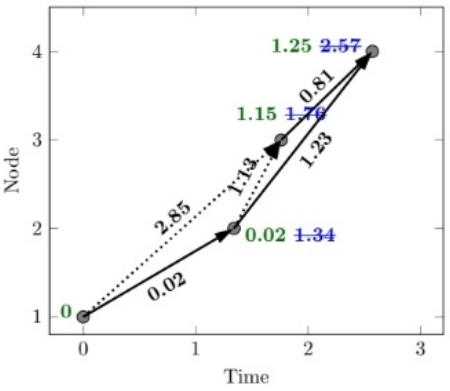
(a) TDSP từ nút $(1, 0)$



(b) BSPT có gốc $(4, 2.57)$



(c) ABSPT $\overline{\mathcal{B}}^{2.57}$



(d) ABSPT $\overline{\mathcal{B}}^{2.57}$ với các UTT khi ghép với $\overline{\mathcal{B}}^{5}$

Hình 4: Quy trình tạo ABSPT ứng với đỉnh xuất phát (1,0)

Giải thích Hình 4:

- Hình 4a: đường đi TDSP từ đỉnh $(1, 0)$ đến n với thời gian là 2.57. Các nút thời gian màu xanh, giá trị hàm vận tốc màu đen.
- Hình 4b: BSPT tương ứng với đỉnh $(4, 2.57)$.
- Hình 4c: ABSPT $\overline{\mathcal{B}}^{2.57}$ tương ứng với đỉnh $(4, 2.57)$. Thời gian di chuyển thực tế có màu đỏ.
- Hình 4d: ABSPT $\overline{\mathcal{B}}^{2.57}$ với các UTT khi ghép với $\overline{\mathcal{B}}^{5}$ (có màu đen), đường đi tối thiểu có màu xanh lá.

Một vài lưu ý khi xây dựng BSPT B^t như sau:

- 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.
- **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 vận tốc $c_{i,j}(t)$ 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:

Định nghĩa 2. *Cây **ABSPT** là một mạng **TEN** được hình thành bằng cách thêm 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 Định nghĩa 2, 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 4c 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 4d 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 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 4d, các nút thuộc đường đi có **UTT** nhỏ nhất trong $\overline{\mathcal{B}}^{2.57}$ được tô màu xanh. Nhãn của nút đích là 1.25, nghĩa là không có đường dẫn hợp lệ nào khác đến nút đích 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 nguồn và đích 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 vận tốc. 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.

1.3 Thời điểm dừng (BP)

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:

TH1: $\exists (i, j) \in A \mid \tau_i : t_i < \tau_i < t_i^+$ với τ_i là BP.
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 thỏa 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^+.

TH2: $\nexists (i, j) \in A \mid \tau_i : t_i < \tau_i < t_i^+$, không có BP 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.

1.4 Thuật toán

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 Thuật toán 1 thực chất là một họ các thuật toán nên không đề cập đến phương án chọn BP khi có nhiều lựa chọn. Chương 2.4 sẽ đề cập đến các cách chọn được sử dụng.

Độ phức tạp thuật toán sẽ là $\mathcal{O}(K \times SSP)$ với K là tổng số BP khám phá được và SSP là độ phức tạp của thuật toán tìm đường đi ngắn nhất từ một đỉnh.

1.4.1 Mã giả

Thuật toán 1 DDD cho bài toán 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 $L \leftarrow (\overline{\mathcal{B}}^{t_0}, \overline{\mathcal{B}}^{T})$;

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

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

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 $\overline{\mathcal{B}}^{t}$ and $\overline{\mathcal{B}}^{t^+}$ **then**

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

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

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

      $UB \leftarrow UB^s$

    end if

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

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

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

  else

    The status of $\overline{\mathcal{B}}^{t}$ 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: $t^+ \leftarrow min\{\tau : \overline{\mathcal{B}}^{\tau}$ is in L and $\tau > t\}$;

end while

1.4.2 Minh họa thuật toán

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

- $\overline{\mathcal{B}}^{2.57}$: 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 BP $\tau = 1 \in [0, 2.90]$ của cung $(1, 2)$.
2. Tìm thấy BP 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 5b, có 3 ABSPT).
4. Cập nhật cận trên và cận dưới: $UB^{3.08} = 2.08$ và $LB^{3.08} = 1.4$.
5. Tính lại và cập nhật cận dưới cho $\overline{\mathcal{B}}^{2.57}$: $LB^{2.57} = 1.89$.

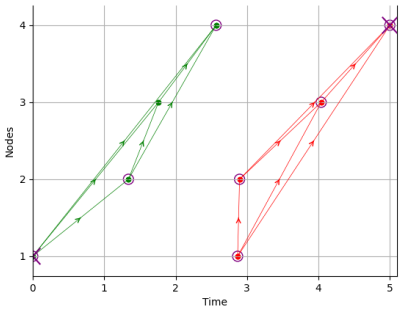
Hiện tại thì $LB = 1.4$ và $UB = 2.08$. Vì $LB < UB$ nên tiếp tục lặp.

Các lần lặp tiếp theo:

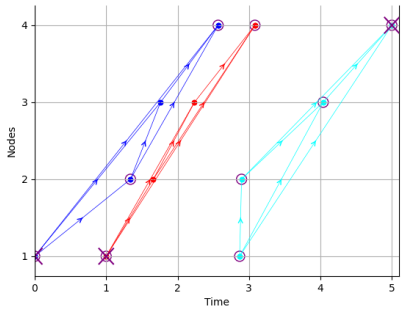
1. Thêm ABSPT $\overline{\mathcal{B}}^{3.89}$ và $\overline{\mathcal{B}}^{3.63}$ tương ứng với $(1, 2)$ và $(2, 2)$. Cái thứ hai cũng là ABSPT thứ ba trong danh sách tại Lặp 4.
2. Cập nhật cận dưới hiện tại: $LB = 1.71$ (từ $\overline{\mathcal{B}}^{3.08}$).
3. Do không có BP giữa **ABSPT** thứ 2 và thứ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

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

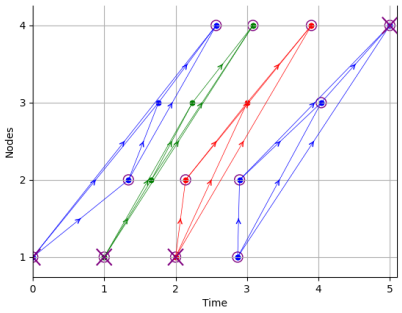
$$((1, 2.00), (2, 2.14), (4, 3.89))$$



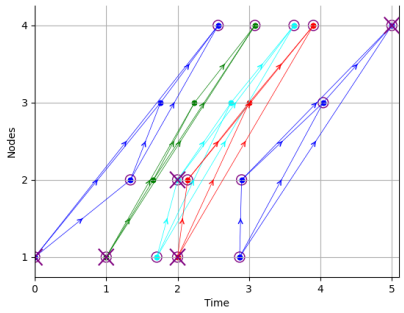
(a) Lần lặp 1



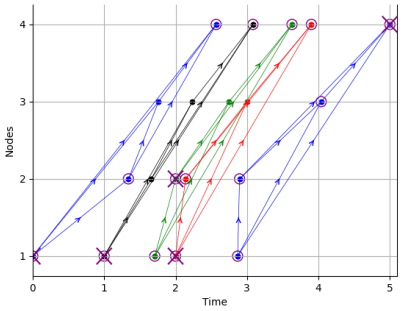
(b) Lần lặp 2



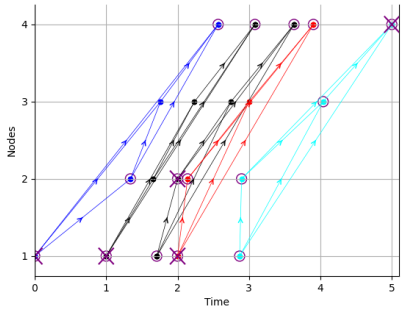
(c) Lần lặp 3



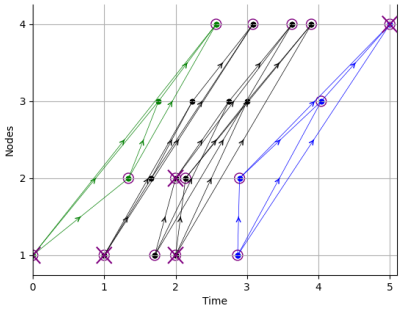
(d) Lần lặp 4



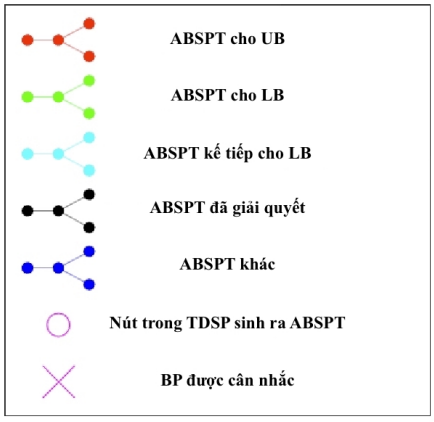
(e) Lần lặp 5



(f) Lần lặp 6



(g) Lần lặp 7



(h) Chú thích

Hình 5: Minh họa cách thức hoạt động của thuật toán trên ví dụ từ Hình 2 và Hình 1.

2 Các thử nghiệm

Để tiến hành đánh giá hiệu năng thuật toán của các chương trước, khóa luận dùng bộ dữ liệu được sinh ngẫu nhiên. Chi tiết về cách tạo dữ liệu được trình bày dưới đây. 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. Việc chạy được thực hiện với các flag tối ưu hóa `-O2` và thư viện Boost 1.73.0

2.1 Tập dữ liệu

Dữ liệu được chuẩn bị cho việc chạy các thuật toán. Có 4 kiểu dữ liệu tương ứng với 3 kiểu mạng $D = (N, A)$ và tập các đỉnh $N = \{1, 2, \dots, n\}$;
Kiểu 1: $A = \{(i, j) \in N \times N | i < j\}$;

- Mỗi cặp đỉnh bất kỳ 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: $A = \{(i, i+1) | i = 1, \dots, n-1\} \cup \{(i, j) \in N \times N | i < j+1$ 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 $U_{i,j} \sim U[0, 1]$

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

- 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 $3/n$.

2.2 Hàm vận tốc

Để đánh giá mức độ ảnh hưởng của hàm vận tốc đến hiệu năng của thuật toán, có hai loại hàm được sử dụng. Hình 6 minh họa cho mỗi loại. Hàm vận tốc 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:

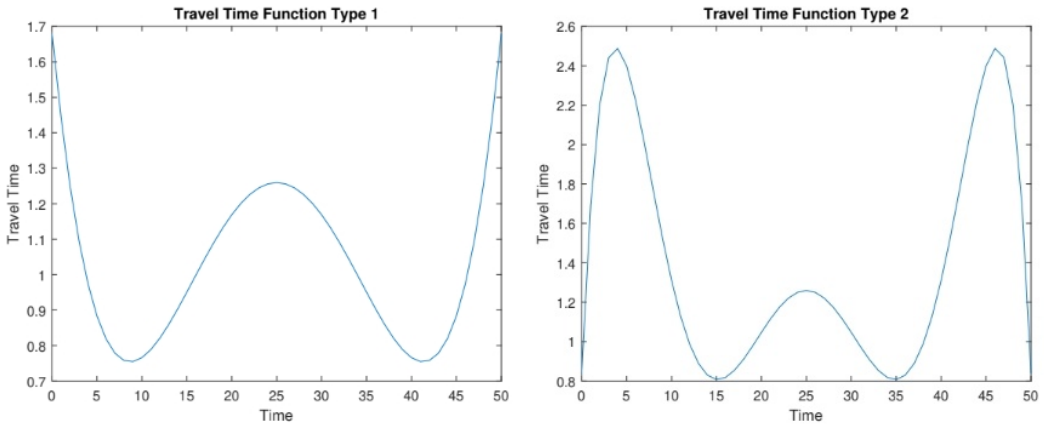
$$f([0, \frac{T}{4}, \frac{T}{2}, \frac{3T}{4}, T]) = \begin{cases} [1.6, 1, 1.05, 1, 1.6](B_{i,j}\frac{|j-i|}{10}), & \text{nếu } U_{i,j} < \frac{1}{3}, \\ [2, 1, 1.5, 1, 2](B_{i,j}\frac{|j-i|}{10}), & \text{nếu } \frac{1}{3} \le U_{i,j} < \frac{2}{3}, \\ [2.5, 1, 1.75, 1, 2.5](B_{i,j}\frac{|j-i|}{10}),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

Với $B_{i,j}, U_{i,j} \sim U[0, 1]$.

Loại 2: f là đa thức bậc 6:

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

Với $B_{i,j}, U_{i,j} \sim U[0, 1]$.



(a) Loại 1 **(b)** Loại 2

Hình 6: Biểu diễn hai loại hàm vận tốc

2.3 Nguồn gốc của dữ liệu

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).

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 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 (hash map) là một cấu trúc dữ liệu hiệu quả để lưu trữ các giá trị của việc tính toán trước khi chạy.

2.4 Kịch bản và cài đặt

Phần này sẽ bàn về ảnh hưởng của việc chọn BP 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 BP 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 BP cho $c_{i,j}(t)$ với $t \in (t_i, t_i^+)$:

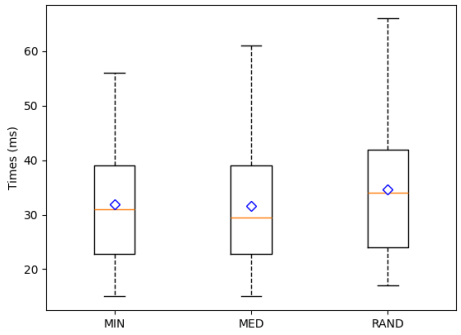
1. Chọn điểm có giá trị $c_{i,j}(t)$ nhỏ nhất với $t \in (t_i, t_i^+)$ (MIN)
2. Chọn điểm trung vị (trong tập các BP sắp xếp theo thời gian) (MED)
3. Chọn một điểm bất kì ($RAND$)

Đánh giá hiệu năng của việc lựa chọn BP:

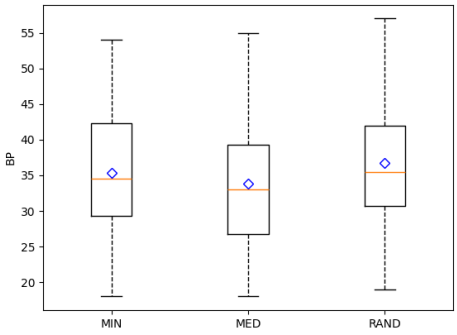
Với bài toán MDP, kết quả được biểu diễn ở Hình 7 bằng biểu đồ hộp với các tỉ số: thời gian chạy, số lần lặp và số BP được giải quyết cho cách chọn MIN, MED và $RAND$.

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

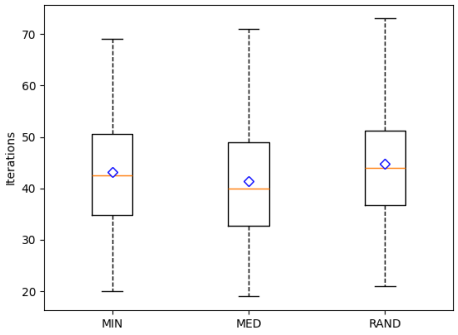
Qua quan sát các yếu tố trung vị, trung bình và hộp, cách chọn MED cho kết quả tốt nhất trong cả ba tiêu chí hiệu năng (thời gian chạy, số lần lặp và số BP được giải quyết). Do đó, MED là lựa chọn ưu tiên trong ba cách.



(a) Thời gian chạy



(b) Số lần lặp



(c) Số BP được giải quyết

Hình 7: So sánh các cách chọn với 60 mẫu của bài toán MDP cho $n = 30, T = 20$

2.5 Kết quả thử nghiệm

2.5.1 Phương pháp liệt kê Enum

Phương pháp DDD sẽ được so sánh với phương pháp Enum liệt kê BP (từ [8]).

Trong bài báo gốc, phương pháp Enum được tóm tắt như sau:

Ý tưởng chính của cách này là chạy thuật toán Dijkstra cho một thời gian khởi hành cụ thể t và xây dựng một tập các *chứng nhận*. Mỗi *chứng nhận* này đảm bảo tính chính xác của cây đường đi ngắn nhất được tìm thấy bởi Dijkstra. Sau đó dần thay đổi thời gian t và chạy Dijkstra lại để tìm ra các *chứng nhận* mới. Bằng cách liên tục điều chỉnh này, phương pháp khám phá một cách có hệ thống cho tất cả các cây đường đi ngắn nhất riêng biệt có thể tồn tại với mối thời gian khởi hành khác nhau. Dẫn tới việc khám phá toàn diện các khả năng dựa trên ràng buộc thời gian. Có hai loại *chứng nhận* được sử dụng: *chứng nhận cơ bản* và *chứng nhận tối thiểu*. *Chứng nhận cơ bản* đảm bảo hàm tuyến tính tương ứng với một đường đi duy nhất không thay đổi cho đến thời điểm thất bại của *chứng nhận*; sự thất bại của một *chứng nhận cơ bản* tương ứng với một BP tại một cạnh nào đó. *Chứng nhận tối thiểu* đảm bảo một đường đi nhất định đến một nút nhất định đầu tiên; sự thất bại của một *chứng nhận tối thiểu* tương ứng với một BP tối thiểu trong đồ thị D. Mỗi *chứng nhận* có một thời điểm thất bại t_{fail}, là thời gian khởi hành sớm nhất sau thời gian hiện tại t mà *chứng nhận* đó không còn hiệu lực.

2.5.2 Kết quả so sánh

Trong phần này khóa luận sẽ so sánh hiệu năng của thuật toán DDD với thuật toán Enum liệt kê BP, với kịch bản chọn là MED.

Bảng 2 trình bày kết quả chạy của bài toán MDP với trung bình 10 mẫu được sinh ngẫu nhiên, có kích thước mạng khác nhau ($n = 30, 50$ và khung thời gian $T = 40$) cùng các loại hàm vận tốc khác nhau.

Bảng 2: Kết quả của MDP với $n = [30, 50]$ và $T = 40$

n	g-type	t-type	Avg. BP	Total BP	% BP	Avg. Arcs	Avg. Time DDD (ms)	Avg. Time Enum (ms)	% Time	Iters
30	1	1	35.2	1133	3.1	5.1	49.7	774.7	6.4	445
		2	44.8	1133	4.0	5.1	66.5	856.2	7.8	528
30	2	1	46.5	1133	4.1	9	32.8	474.9	6.9	584
		2	46.5	1133	4.1	9	34.2	515.4	6.6	596
30	3	1	34.2	1133	3.0	6.7	33.7	621.1	5.4	415
		2	45.6	1133	4.0	6.7	45.7	667.4	6.8	548
50	1	1	45.5	1913	2.4	7.6	115.6	2375.9	4.9	567
		2	52.7	1913	2.8	7.5	126.8	2496.5	5.1	657
50	2	1	53.7	1913	2.8	13	63.7	1352.1	4.7	671
		2	56.2	1913	2.9	13	68.7	1461.1	4.7	690
50	3	1	58.9	1913	3.1	9	116.9	1934.2	6.0	752
		2	49.2	1913	2.6	8.9	98.4	2043.5	4.8	601

Các cột trong bảng có ý nghĩa như sau:

- **n**: số lượng đỉnh trong mạng lưới.
- **g-type**: loại mạng lưới, có ba loại 1,2,3.
- **t-type**: loại hàm vận tốc, có hai loại 1 và 2.
- **Avg. BP**: số BP khám phá được trung bình bằng phương pháp DDD.
- **Total BP**: tổng số BP trong mẫu (phương pháp Enum liệt kê).
- **% BP**: tỉ lệ phần trăm số BP được khám phá.
- **Avg. Arcs**: số lượng cung trung bình trong nghiệm tối ưu.
- **Avg. Time DDD**: thời gian chạy trung bình của phương pháp DDD (đơn vị ms).
- **Avg. Time Enum**: thời gian chạy trung bình của phương pháp Enum (đơn vị ms).
- **% Time**: tỉ lệ phần trăm thời gian chạy của DDD với phương pháp

Enum.

- **Iters**: số lần lặp trung bình của phương pháp DDD.

Lưu ý rằng tổng số BP có thể có trong các mẫu này là $|N-1| \times (T-1)+2$, khác với công thức $|A| \times (T-1)+2$ của [8]. Vì đối với các cung có cùng hai đỉnh đầu cuối, ta chỉ cần xử lý một lần.

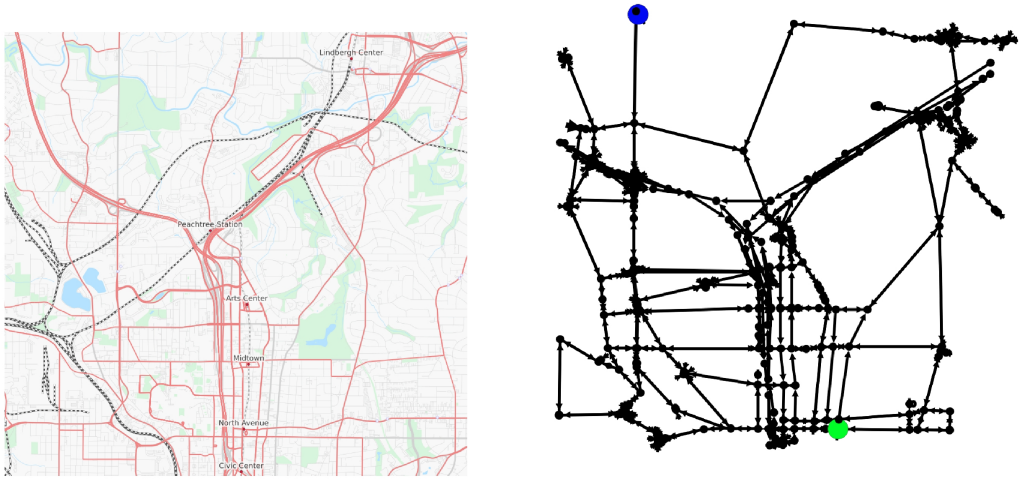
Kết quả cho thấy phương pháp DDD chỉ cần khám phá một phần nhỏ (từ 2.4% đến 4.1%) tổng số BP, được minh họa bằng Hình 11. Điều này cho thấy phương pháp này hoạt động hiệu quả, chỉ tập trung vào các BP tiềm năng thay vì toàn bộ. Ta cũng có thể thấy rằng khi kích cỡ mẫu tăng lên thì lợi ích của phương pháp DDD càng rõ rệt, như ở các mẫu $n = 50$ thì phần trăm BP được khám phá giảm nhiều hơn so với ở các mẫu $n = 30$. Các kết quả chạy với tập mẫu lớn hơn được trình bày ở Bảng 4, với các giá trị **% BP** và **% Time** trung bình giảm dần khi mẫu càng lớn. Ngoài ra thời gian chạy của phương pháp DDD đã cho thấy sự vượt trội so với phương pháp Enum, mô tả ở hình Hình 12.

2.6 Trường hợp thực tế

Ngoài việc thử nghiệm với các dữ liệu sinh ngẫu nhiên, thuật toán DDD cho bài toán MDP đã được thử nghiệm trên bộ dữ liệu thực tế từ mạng lưới đường bộ Atlanta từ bài báo ở [11]. Mạng này bao gồm 306 nút và 618 cung, được lấy từ Open Street Map. Hình 8 mô tả bản đồ khu vực và biểu diễn mạng lưới tương ứng.

Dữ liệu về thời gian di chuyển được thu thập từ nhiều nguồn khác nhau (hệ thống định vị, dữ liệu di động). Dữ liệu được thu thập và tổng hợp để tạo thành các hàm vận tốc tuyến tính từng khúc với các BP cách nhau 15 phút trong 24 giờ. Nhờ vậy mỗi cung có tổng cộng 192 điểm dữ liệu. Hình 9 là ví dụ về các hàm vận tốc này.

Để minh họa sự khác biệt giữa đường đi tối thiểu thời gian thực hiện, đến đích sớm nhất và khởi hành muộn nhất từ nguồn đến đích trong khoảng thời



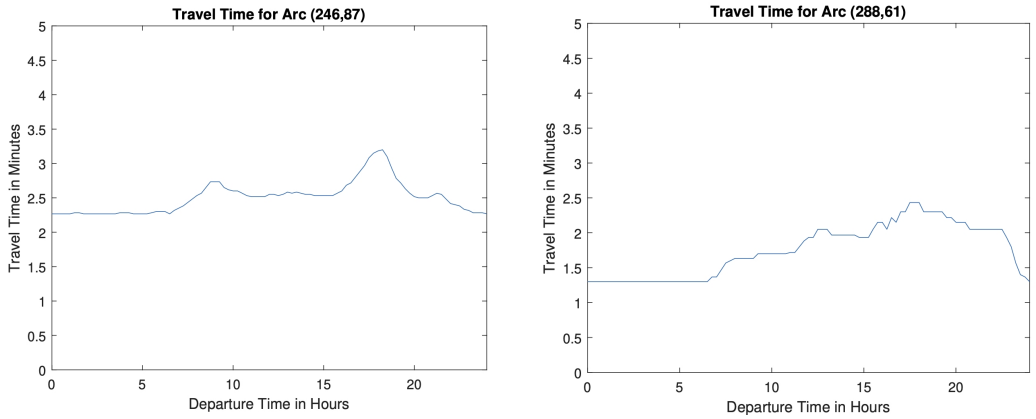
(a) Bản đồ khu vực **(b)** Mạng lưới đường bộ

Hình 8: Bản đồ và mạng lưới đường bộ khu vực Trung tâm và phía Bắc Atlanta

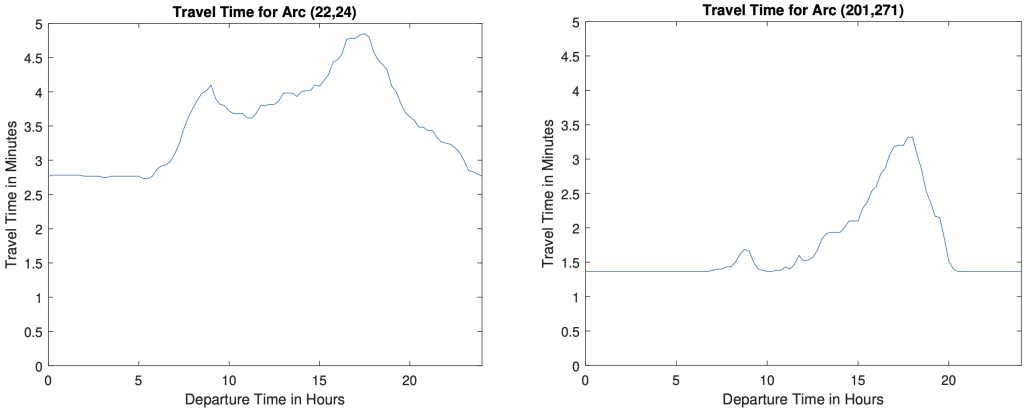
gian cụ thể, tôi chọn điểm nguồn ở Bắc Atlanta, đích đến ở Trung tâm Atlanta và khung thời gian là $[9:30, 13:00]$.

Đường đi đến đích sớm nhất mất 47.5 phút còn đường đi xuất phát muộn nhất mất 47.8 phút (Hình 10). Hai đường đi này đi đường qua đường tránh cao tốc, qua khu dân cư (đường Peachtree và đường Juniper) và phải đi đường vòng để tránh một chiều.

Mặc dù sự khác biệt trong thử nghiệm này không quá lớn (do khu vực Atlanta được chọn khá bé và ít trường học), việc tìm đường đi chỉ mất vài giây thời gian chạy. Điều này cho thấy phương pháp này có thể hữu ích trong thực tế.

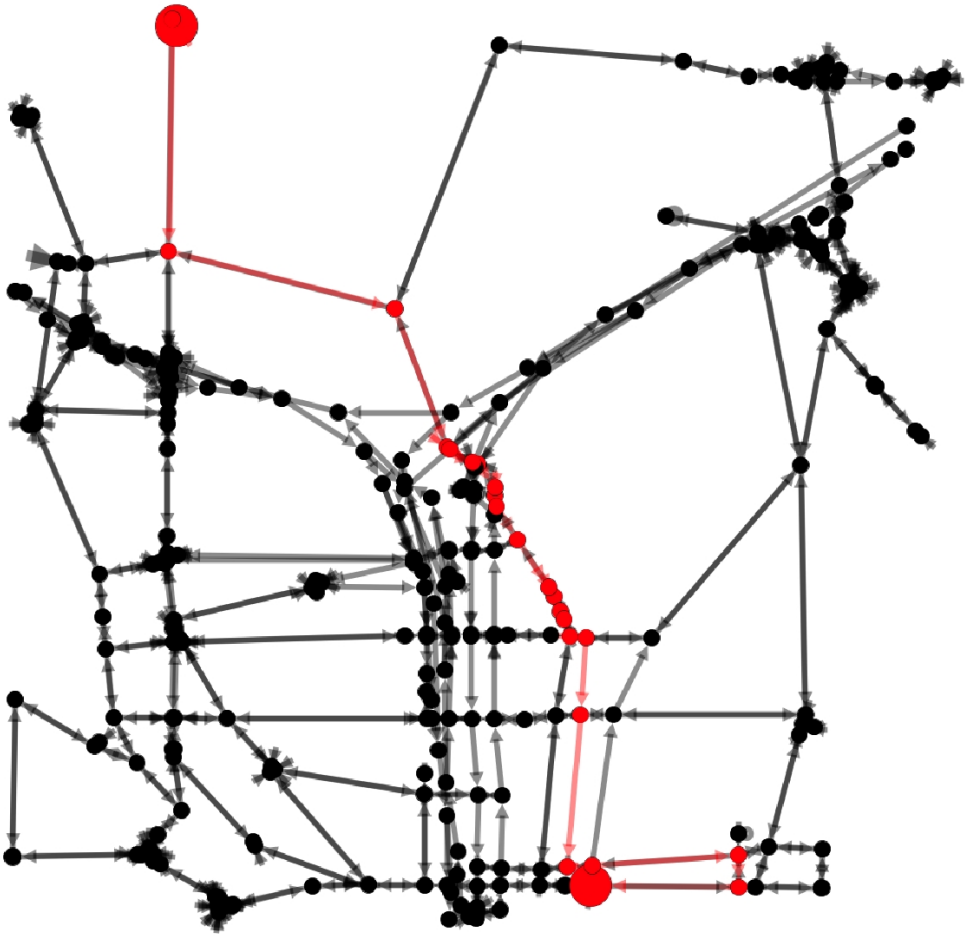


(a) Đường Peachtree **(b)** Đại lộ Ponce de Leon NE

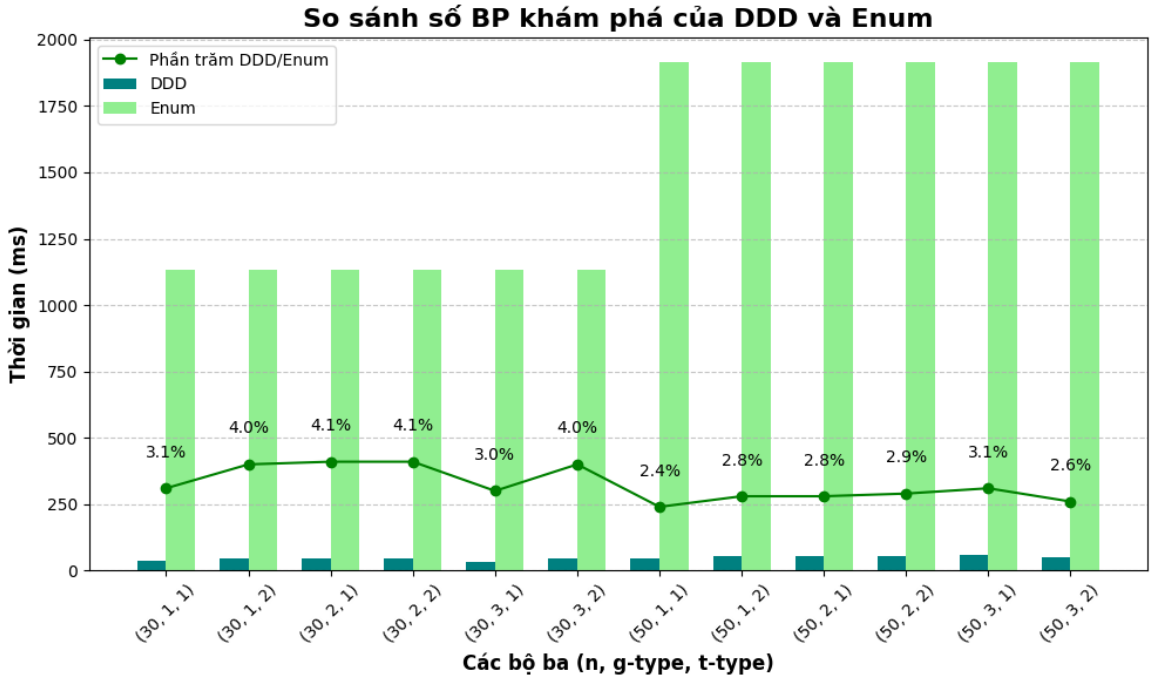


(c) Bắc Ave. NE **(d)** Trung tâm khu vực I-75/85

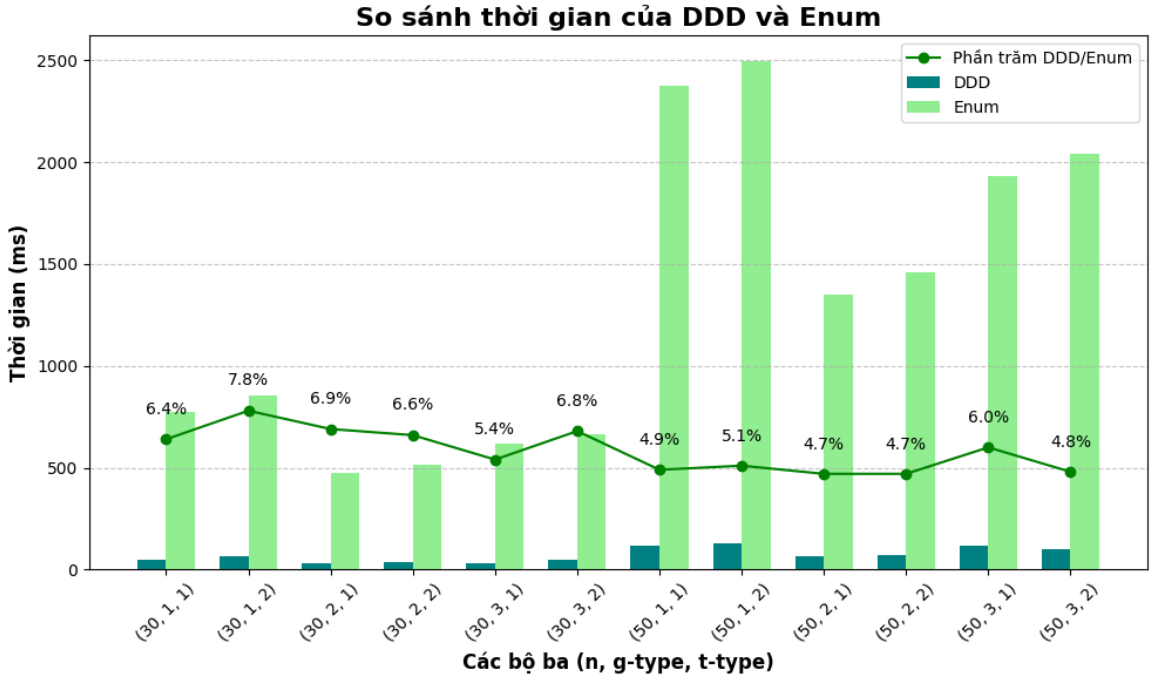
Hình 9: Hàm vận tốc các khu vực ở Atlanta



Hình 10: Đường đi từ Bắc Atlanta đến Trung tâm Atlanta, đến đích sớm nhất / khởi hành muộn nhất



Hình 11: So sánh số BP của $n = [30, 50]$ và $T = 40$



Hình 12: So sánh thời gian chạy của $n = [30, 50]$ và $T = 40$

Kết luận

Khoá luận đã trình bày về thuật toán khám phá rời rạc động cho bài toán tìm đường có thời gian di chuyển tối thiểu, trong đó thời gian di chuyển trên cung là hàm vận tốc tuyến tính từng khúc của thời gian khởi hành trên cung đó.

Thuật toán này tận dụng các BP của mỗi hàm vận tốc và các giá trị trước đó cho việc giải bài toán thời gian tối thiểu. Ưu điểm vượt trội của thuật toán là việc chỉ cần sử dụng một phần nhỏ các BP của hàm vận tốc để tìm nghiệm và khẳng định tính tối ưu của nó. Bài toán khoá luận này nghiên cứu là bài toán tìm đường đi có thời gian di chuyển tối thiểu (MDP), tức là bài toán tìm đường đi có thời gian di chuyển ít nhất với hàm vận tốc phụ thuộc thời điểm. Bài toán này chỉ cho phép việc chờ đợi ở nút nguồn.

Kết quả của khóa luận đã cho thấy thuật toán DDD có thể giải quyết bài toán đề ra với kết quả vượt trội phương pháp Enum, thời gian chạy nhanh. Đây là tiền đề để ứng dụng trong các bài toán thực tế. Tuy nhiên vẫn còn hai hạn chế chính của bài luận này. Thứ nhất, thuật toán đã đơn giản hóa các ràng buộc ngoại lai như thời gian chờ đợi và chi phí di chuyển khác giữa các nút, điều này làm giảm tính thực tế của bài toán. Thứ hai là các dữ liệu sử dụng để thử nghiệm vẫn còn khá nhỏ, chưa đánh giá được hiệu suất của thuật toán với các bộ lớn hơn.

Mã nguồn của chương trình cài đặt thuật toán sử dụng ngôn ngữ C++17, các flag tối ưu (-O2) và thư viện Boost được sử dụng trong quá trình biên dịch để tăng tốc độ chạy. Kết quả thử nghiệm cho thấy hiệu suất tăng từ 300% đến 500% so với khi không thêm tối ưu.

Dựa vào những điều đã kết luận ở trên, đây là những hướng phát triển cho

tương lai:

- Nghiên cứu thêm bài toán các nút có thể chờ đợi trong mạng.
- Cài đặt các thuật toán tìm đường đi ngắn nhất khác như A*.
- Sử dụng với các bộ dữ liệu thực tế để đánh giá hiệu quả thuật toán.

Tài liệu tham khảo

[1] R. Bellman. “On a routing problem”. *Quarterly of applied mathematics* 16.1 (1958), 87–90.

[2] N. Boland, M. Hewitt, L. Marshall & M. Savelsbergh. “The continuous-time service network design problem”. *Operations research* 65.5 (2017), 1303–1321.

[3] K. L. Cooke & E. Halsey. “The shortest route through a network with time-dependent internodal transit times”. *Journal of mathematical analysis and applications* 14.3 (1966), 493–498.

[4] B. C. Dean. “Shortest paths in FIFO time-dependent networks: Theory and algorithms”. *Rapport technique, Massachusetts Institute of Technology* 13 (2004).

[5] U. Demiryurek, F. Banaei-Kashani, C. Shahabi & A. Ranganathan. “Online computation of fastest path in time-dependent spatial networks”. *International Symposium on Spatial and Temporal Databases*. Springer. 2011, 92–111.

[6] M. A. Figliozzi. “The time dependent vehicle routing problem with time windows: Benchmark problems, an efficient solution algorithm, and solution characteristics”. *Transportation Research Part E: Logistics and Transportation Review* 48.3 (2012), 616–636.

[7] L. Ford & D. Fulkerson. **Flows in Networks**. Princeton landmarks in mathematics and physics. Princeton University Press, 2010. isbn: 9780691146676. url: https://books.google.com.vn/books?id=9h-DQgAACAAJ.

[8] L. Foschini, J. Hershberger & S. Suri. “On the complexity of time-dependent shortest paths”. *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2011, 327–341.

[9] V. Gunturi, K. Joseph, S. Shekhar & K. M. Carley. “Information lifetime aware analysis for dynamic social networks” (2012).

[10] E. He, N. Boland, G. Nemhauser & M. Savelsbergh. “A dynamic discretization discovery algorithm for the minimum duration time-dependent shortest path problem”. *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26–29, 2018, Proceedings 15*. Springer. 2018, 289–297.

[11] E. Y. He, N. Boland, G. Nemhauser & M. Savelsbergh. “Dynamic discretization discovery algorithms for time-dependent shortest path problems”. *INFORMS Journal on Computing* 34.2 (2022), 1086–1114.

[12] A. Orda & R. Rom. “Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length”. *Journal of the ACM (JACM)* 37.3 (1990), 607–625.

Phụ lục

2.7 Các hàm cho ví dụ trong Hình 3

Hàm vận tốc cho ví dụ trong Hình 3 như sau:

$$c_{1,2}(t) = \begin{cases} 1.34 - 0.68t, & 0 \le t < 1 \\ 1.18 - 0.52t, & 1 \le t < 2 \\ 0.40 - 0.13t, & 2 \le t < 3 \\ -1.01 + 0.34t, & 3 \le t < 4 \\ -2.25 + 0.65t, & 4 \le t < 5 \end{cases} \qquad c_{1,2}^{rev}(t) = \begin{cases} 1.34 - 2.13t, & 1.34 \le t < 1.66 \\ 0.66 - 1.08t, & 1.66 \le t < 2.14 \\ 0.14 - 0.15t, & 2.14 \le t < 3.01 \\ 0.01 + 0.25t, & 3.01 \le t < 4.35 \\ 0.35 + 0.39t, & 4.35 \le t < 6.00 \end{cases}$$

$$c_{1,3}(t) = \begin{cases} 2.85 + 0.10t, & 0 \le t < 1 \\ 2.90 + 0.05t, & 1 \le t < 2 \\ 3.04 - 0.02t, & 2 \le t < 3 \\ 3.22 - 0.08t, & 3 \le t < 4 \\ 3.46 - 0.14t, & 4 \le t < 5 \end{cases} \qquad c_{1,3}^{rev}(t) = \begin{cases} 2.85 + 0.09t, & 2.85 \le t < 3.95 \\ 2.95 + 0.05t, & 3.95 \le t < 5.00 \\ 3.00 - 0.02t, & 5.00 \le t < 5.98 \\ 2.98 - 0.09t, & 5.98 \le t < 6.90 \\ 2.90 - 0.16t, & 6.90 \le t < 7.76 \end{cases}$$

$$c_{2,3}(t) = \begin{cases} 1.99 - 0.17t, & 0 \le t < 1 \\ 2.13 - 0.31t, & 1 \le t < 2 \\ 2.33 - 0.41t, & 2 \le t < 3 \\ 2.39 - 0.43t, & 3 \le t < 4 \\ 2.15 - 0.37t, & 4 \le t < 5 \end{cases} \qquad c_{2,3}^{rev}(t) = \begin{cases} 1.99 - 0.20t, & 1.99 \le t < 2.82 \\ 1.82 - 0.45t, & 2.82 \le t < 3.51 \\ 1.51 - 0.69t, & 3.51 \le t < 4.10 \\ 1.10 - 0.75t, & 4.10 \le t < 4.67 \\ 0.67 - 0.59t, & 4.67 \le t < 5.30 \end{cases}$$

$$c_{2,4}(t) = \begin{cases} 1.29 - 0.27t, & 0 \le t < 1 \\ 0.41 + 0.61t, & 1 \le t < 2 \\ -0.25 + 0.94t, & 2 \le t < 3 \\ 1.28 + 0.43t, & 3 \le t < 4 \\ 4.84 - 0.46t, & 4 \le t < 5 \end{cases} \qquad c_{2,4}^{rev}(t) = \begin{cases} 1.29 - 0.37t, & 1.29 \le t < 2.02 \\ 1.02 + 0.38t, & 2.02 \le t < 3.63 \\ 1.63 + 0.48t, & 3.63 \le t < 5.57 \\ 2.57 + 0.30t, & 5.57 \le t < 7.00 \\ 3.00 - 0.95t, & 7.00 \le t < 7.54 \end{cases}$$

$$c_{3,4}(t) = \begin{cases} 0.61 + 0.12t, & 0 \le t < 1 \\ 0.63 + 0.10t, & 1 \le t < 2 \\ 0.72 + 0.06t, & 2 \le t < 5 \end{cases} \qquad c_{3,4}^{rev}(t) = \begin{cases} 0.61 + 0.11t, & 0.61 \le t < 1.73 \\ 0.73 + 0.09t, & 1.73 \le t < 2.83 \\ 0.93 + 0.05t, & 2.83 \le t < 6.00 \end{cases}$$

Bảng 3: Bảng hàm ngược thời gian di chuyển cho ví dụ trong Hình 3

BP	(1,2)	BP	(1,3)	BP	(2,3)	BP	(2,4)	BP	(3,4)
1.34	1.34	2.85	2.85	1.99	1.99	1.29	1.29	0.61	0.61
1.66	0.66	3.95	2.95	2.82	1.82	2.02	1.02	1.73	0.73
2.14	0.14	5	3	3.51	1.51	3.63	1.63	2.83	0.93
3.01	0.01	5.98	2.98	4.1	1.1	5.57	2.57	—	—
4.35	0.35	6.9	2.9	4.67	0.67	7	3	—	—
6	1	7.76	2.76	5.3	0.3	7.54	2.54	6	1

2.8 Toàn bộ kết quả chạy của MDP

Bảng 4: MDP với $n = [30, 50]$ và $T = [20, 40, 60, 80, 100]$

n	T	g-type	t-type	Avg. BP	Total BP	% BP	Avg. Arcs	Avg. DDD Time (ms)	Avg Enum Time (ms)	% Time	Iters
30	20	1	1	32.6	553	5.9	5.1	38.1	348	10.9	398
30	20	1	2	32.6	553	5.9	5.1	40.3	378.9	10.6	393
30	20	2	1	36.8	553	6.7	9	22.5	212.8	10.6	470
30	20	2	2	38.8	553	7.0	9	23.3	231.6	10.1	487
30	20	3	1	29.9	553	5.4	6.8	28.4	296.4	9.6	351
30	20	3	2	32.6	553	5.9	6.7	30.9	321.9	9.6	389
30	40	1	1	35.2	1133	3.1	5.1	41.6	686.8	6.1	445
30	40	1	2	44.8	1133	4.0	5.1	51.9	738	7.0	528
30	40	2	1	46.5	1133	4.1	9	28.8	431.3	6.7	584
30	40	2	2	46.5	1133	4.1	9	29.4	484.8	6.1	596
30	40	3	1	34.2	1133	3.0	6.7	32.5	590.5	5.5	415
30	40	3	2	45.6	1133	4.0	6.7	44.5	652.5	6.8	548
30	60	1	1	48.8	1713	2.8	5.2	58.2	1025	5.7	303
30	60	1	2	48.2	1713	2.8	5.2	60.2	1100.2	5.5	281

Bảng 4: MDP với $n = [30, 50]$ và $T = [20, 40, 60, 80, 100]$

n	T	g-type	t-type	Avg. BP	Total BP	% BP	Avg. Arcs	Avg. DDD Time (ms)	Avg Enum Time (ms)	% Time	Iters
30	60	2	1	50.4	1713	2.9	9	30.8	644.8	4.8	311
30	60	2	2	42.8	1713	2.5	9	26.4	709.6	3.7	278
30	60	3	1	54.8	1713	3.2	7.4	58.6	905.8	6.5	350
30	60	3	2	49.2	1713	2.9	7.2	49.2	978	5.0	322
30	100	1	1	61.4	2873	2.1	5.2	84.4	1957.6	4.3	392
30	100	1	2	58	2873	2.0	5.2	73.2	2002.6	3.7	357
30	100	2	1	52.6	2873	1.8	9	36.2	1252	2.9	332
30	100	2	2	58.6	2873	2.0	9	41.2	1263.2	3.3	360
30	100	3	1	57.8	2873	2.0	7.4	64	1725	3.7	359
30	100	3	2	76.6	2873	2.7	7.2	82.2	1721	4.8	472
50	20	1	1	36.9	933	4.0	7.5	83.2	1135.7	7.3	440
50	20	1	2	49.2	933	5.3	7.5	117.3	1252.6	9.4	584
50	20	2	1	47.6	933	5.1	13	55.1	651.5	8.5	590
50	20	2	2	46.6	933	5.0	13	53	700.2	7.6	562

Bảng 4: MDP với $n = [30, 50]$ và $T = [20, 40, 60, 80, 100]$

n	T	g-type	t-type	Avg. BP	Total BP	% BP	Avg. Arcs	Avg. DDD Time (ms)	Avg Enum Time (ms)	% Time	Iters
50	20	3	1	47	933	5.0	9	87.8	919.6	9.5	589
50	20	3	2	41.2	933	4.4	8.9	76	993.4	7.7	478
50	40	1	1	45.5	1913	2.4	7.6	103.2	2262.9	4.6	567
50	40	1	2	52.7	1913	2.8	7.5	122	2483.7	4.9	657
50	40	2	1	53.7	1913	2.8	13	86.3	1852.1	4.7	671
50	40	2	2	56.2	1913	2.9	13	64.1	1425.9	4.5	690
50	40	3	1	58.9	1913	3.1	9	156.5	2578.5	6.1	752
50	40	3	2	49.2	1913	2.6	8.9	95.7	2033.5	4.7	601
50	60	1	1	56.8	2893	2.0	7.6	149.6	3914.9	3.8	719
50	60	1	2	57.1	2893	2.0	7.5	157.8	4273.1	3.7	724
50	60	2	1	69.5	2893	2.4	13	103.4	2444.2	4.2	874
50	60	2	2	66.5	2893	2.3	13	77.3	2132.3	3.6	831
50	60	3	1	64	2893	2.2	9	133.3	3009.6	4.4	807
50	60	3	2	58	2893	2.0	8.9	131.7	3386.5	3.9	740

Bảng 4: MDP với $n = [30, 50]$ và $T = [20, 40, 60, 80, 100]$

n	T	g-type	t-type	Avg. BP	Total BP	% BP	Avg. Arcs	Avg. DDD Time (ms)	Avg Enum Time (ms)	% Time	Iters
50	80	1	1	63.6	3873	1.6	7.6	190.2	5586.9	3.4	818
50	80	1	2	61.5	3873	1.6	7.5	152.7	5618.3	2.7	750
50	80	2	1	74.7	3873	1.9	13	110.5	3164.3	3.5	946
50	80	2	2	71.2	3873	1.8	13	91.5	3334.3	2.7	904
50	80	3	1	74.8	3873	1.9	8.9	164.7	4402.5	3.7	940
50	80	3	2	68.6	3873	1.8	8.9	152.1	4553.7	3.3	860
50	100	1	1	67.3	4853	1.4	7.6	179.7	6127	2.9	844
50	100	1	2	70.6	4853	1.5	7.5	182.2	6775	2.7	889
50	100	2	1	82	4853	1.7	13	98.1	3290.7	3.0	1058
50	100	2	2	81.2	4853	1.7	13	95.4	3579.5	2.7	1031
50	100	3	1	86.3	4853	1.8	9	167.2	4685.1	3.6	1052
50	100	3	2	71.9	4853	1.5	8.9	152.8	6098.2	2.5	903

Những phần đã chỉnh sửa

Những nội dung đã chỉnh sửa trong khóa luận theo nhận xét của cán bộ phản biện và hội đồng:

- Sửa các câu văn trong phần mở đầu; bổ sung thêm các thông tin cần thiết về bài toán.
- Sửa các lỗi chính tả, đánh số lại các hình và bảng (theo số bắt đầu từ 1).
- Thay các hình bị mờ trong phần Minh họa thuật toán.
- Thêm phần mô tả về phương pháp Enum kèm trích dẫn ở mục 2.5.1
- Làm rõ các hàm mục tiêu: Sửa từ ”Tối thiểu thời gian thực hiện (đi từ nguồn đến đích)” thành ”Tìm một đường đi có thời gian thực hiện tối thiểu (từ nguồn đến đích)”
- Làm rõ đầu vào và đầu ra cho bài toán MDP ở ngay đầu chương 1.
- Sửa các trích dẫn trong phần Tham khảo.