ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA TOÁN - CƠ - TIN HỌC

Nghiêm Quốc Hưng

BÀI TOÁN TÁI CẤU HÌNH TẬP ĐỘC LẬP TRÊN ĐỒ THỊ CÓ HƯỚNG

Khóa luận tốt nghiệp đại học hệ chính quy
Ngành: **Máy tính và Khoa học thông tin**
(Chương trình đào tạo chất lượng cao)

Hà Nội - 2025

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA TOÁN - CƠ - TIN HỌC

Nghiêm Quốc Hưng

BÀI TOÁN TÁI CẤU HÌNH TẬP ĐỘC LẬP TRÊN ĐỒ THỊ CÓ HƯỚNG

Khóa luận tốt nghiệp đại học hệ chính quy
Ngành: **Máy tính và Khoa học thông tin**
(Chương trình đào tạo chất lượng cao)

Cán bộ hướng dẫn: TS. Hoàng Anh Đức

Hà Nội - 2025

Lời cảm ơn

Lời đầu tiên, em xin bày tỏ lòng biết ơn sâu sắc tới TS. Hoàng Anh Đức, người đã trực tiếp hướng dẫn và đồng hành cùng em trong suốt quá trình thực hiện khóa luận tốt nghiệp. Thầy luôn tận tình chỉ dẫn, giải đáp mọi thắc mắc và đưa ra những góp ý quý báu, sát sao, giúp em định hướng rõ ràng trong từng bước triển khai đề tài. Nhờ sự tâm huyết và tận tụy của thầy, em đã có thể hoàn thành khóa luận một cách trọn vẹn nhất.

Em cũng xin gửi lời cảm ơn chân thành đến Ban giám hiệu, các thầy cô giảng viên, cán bộ và nhân viên Trường Đại học Khoa học Tự nhiên, Đại học Quốc gia Hà Nội – những người đã tạo nên một môi trường học tập năng động, giàu cảm hứng và đầy cơ hội trong suốt bốn năm qua. Đặc biệt, em xin tri ân các thầy cô trong Khoa Toán – Cơ – Tin học đã trực tiếp giảng dạy, truyền đạt cho em nhiều kiến thức quý báu và kỹ năng chuyên môn vững chắc.

Bên cạnh đó, em cũng xin cảm ơn các anh chị khóa trên, các bạn đồng môn và các em khóa dưới đã hỗ trợ, chia sẻ và đồng hành cùng em trong quá trình học tập và nghiên cứu.

Cuối cùng, em xin dành lời cảm ơn sâu sắc nhất tới gia đình, những người luôn là chỗ dựa tinh thần vững chắc, luôn tin tưởng, động viên và tiếp thêm nghị lực cho em trong mọi chặng đường.

Trong quá trình thực hiện khóa luận, dù đã rất nỗ lực, song do hạn chế về thời gian và kinh nghiệm, em không tránh khỏi những thiếu sót. Em kính mong nhận được sự góp ý từ các thầy cô trong Hội đồng để hoàn thiện hơn đề tài này trong tương lai.

Em xin chân thành cảm ơn!

Nghiêm Quốc Hưng

Danh mục viết tắt và thuật ngữ chuyên môn

Viết tắt	Tiếng Anh	Tiếng Việt
DAG	Directed Acyclic Graph	Đồ thị có hướng không chu trình
DPM	Directed Path Matching	Phép ghép các đường đi có hướng
DTS	Directed Token Sliding	Bài toán trượt token có hướng
MIS	Multicolored Independent Set	Tập độc lập đa sắc
TS	Token Sliding	Trượt token (trên đồ thị vô hướng)
	Biased Path Pair	Cặp đường đi lệch hướng
	Path-set Conditions	Các điều kiện của tập đường đi
	Polytree	Đa cây có hướng
	Token Dependency Graph	Đồ thị phụ thuộc giữa các token
	Underlying Undirected Graph	Đồ thị vô hướng nền
	Weakly Biased Path Pair	Cặp đường đi lệch hướng yếu

Mục lục

Danh sách hình vẽ

Danh sách bảng

LỜI NÓI ĐẦU

Trong thế giới ngày càng phát triển của khoa học máy tính và tối ưu hóa tổ hợp, bài toán tái cấu hình tập hợp độc lập trên đồ thị có hướng xuất hiện như một vấn đề quan trọng, vừa mang tính lý thuyết vừa có nhiều ứng dụng thực tế. Bài toán tái cấu hình, có thể hiểu đơn giản là quá trình biến đổi một trạng thái hợp lệ thành một trạng thái hợp lệ khác thông qua một chuỗi các bước trung gian mà vẫn đảm bảo các quy tắc nhất định. Trong bài toán này, trạng thái chính là tập hợp các đỉnh độc lập trên đồ thị có hướng, và quá trình tái cấu hình yêu cầu tìm một dãy di chuyển sao cho tập hợp ban đầu có thể chuyển đổi thành tập hợp đích theo cách hợp lệ.

Vậy tại sao bài toán này lại quan trọng đến vậy? Hãy thử tưởng tượng một hệ thống mạng lưới robot tự hành cần thay đổi vị trí của từng robot mà không gây xung đột, hoặc một hệ thống quản lý tài nguyên điện toán đám mây cần điều phối lại các tiến trình sao cho không có hai tiến trình quan trọng nào bị đặt chung trên một tài nguyên giới hạn. Những vấn đề như vậy không chỉ đơn thuần là bài toán lý thuyết, mà còn xuất hiện trong thực tế và cần những giải pháp hiệu quả.

Trên đồ thị vô hướng, bài toán tái cấu hình tập hợp độc lập đã được nghiên cứu khá kỹ lưỡng. Tuy nhiên, khi mở rộng sang đồ thị có hướng, mọi thứ trở nên phức tạp hơn rất nhiều. Nếu như trên đồ thị vô hướng, một token (đại diện cho một phần tử trong tập hợp) có thể di chuyển tự do giữa hai đỉnh kề nhau, thì trên đồ thị có hướng, token chỉ có thể di chuyển theo chiều của các cung. Điều này làm thay đổi hoàn toàn cách tiếp cận khi kiểm tra tính khả thi của một chuỗi tái cấu hình và đòi hỏi những phương pháp mới để tìm ra lời giải tối ưu.

Mục tiêu của khóa luận này là trình bày và phân tích lại các kết quả trong bài báo *"Independent Set Reconfiguration on Directed Graphs"* của các tác giả Takehiro Ito, Yuni Iwamasa, Yasuaki Kobayashi, Yu Nakahata, Yota Otachi, Masahiro Takahashi và Kunihiro Wasa (2022) [1]. Khóa luận sẽ tập trung vào việc giới thiệu điều kiện cần và đủ để xác định khi nào một tập hợp độc lập có thể tái cấu hình thành một tập hợp khác, cũng như thuật toán kiểm tra và xây dựng chuỗi tái cấu hình hiệu quả. Đặc biệt, khóa luận làm rõ các kết quả trên lớp đồ thị polytree (đa cây có hướng), nhằm giúp

người đọc hiểu sâu hơn về bản chất của bài toán và đánh giá được độ phức tạp tính toán trong từng trường hợp.

Ngoài ý nghĩa về mặt lý thuyết, bài toán này còn có nhiều ứng dụng thực tế đáng chú ý. Trong các hệ thống phân tán, việc di chuyển các tiến trình mà không làm gián đoạn hoạt động là một bài toán quan trọng. Trong lĩnh vực trí tuệ nhân tạo, đặc biệt là lập kế hoạch chuyển động (motion planning), thuật toán tái cấu hình có thể giúp tìm kiếm các chiến lược di chuyển tối ưu trong không gian có ràng buộc. Ngoài ra, các bài toán như quản lý luồng công việc, kiểm soát truy cập, tối ưu hóa năng lượng trong hệ thống cảm biến cũng có thể hưởng lợi từ cách tiếp cận này.

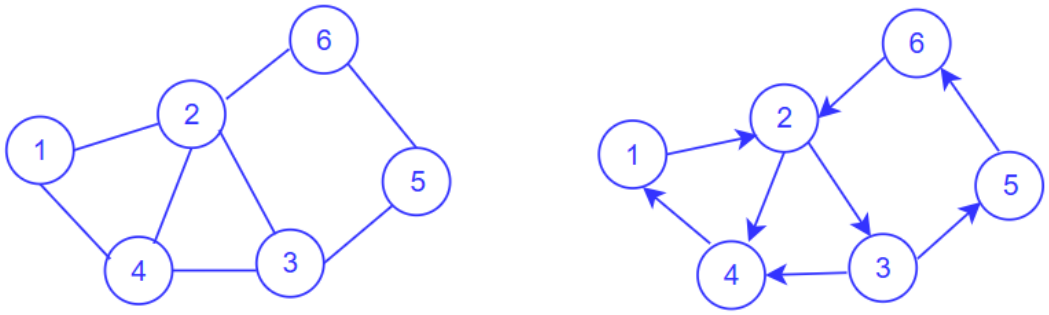
Nhìn chung, bài toán tái cấu hình tập hợp độc lập trên đồ thị có hướng không chỉ mang lại nhiều thách thức khoa học thú vị mà còn mở ra nhiều cơ hội ứng dụng thực tiễn. Việc tìm ra các thuật toán hiệu quả và hiểu rõ điều kiện đảm bảo khả năng tái cấu hình sẽ giúp mở rộng phạm vi ứng dụng của bài toán này trong nhiều lĩnh vực quan trọng của cuộc sống.

Chương 1

KIẾN THỨC CƠ SỞ

1.1 Tổng quan về lý thuyết đồ thị

Chúng ta định nghĩa một đồ thị $G = (V, E)$, trong đó V là một tập hợp hữu hạn các đỉnh và E là tập hợp các cạnh. Mỗi cạnh của đồ thị là một cặp đỉnh phân biệt được biểu diễn dưới dạng một cặp giá trị $\{u, v\}$ (có thể được viết thành uv) . Về cơ bản đồ thị là một tập hợp hữu hạn gồm các đỉnh và được nối với nhau bởi các cạnh.



(a) Đồ thị vô hướng (b) Đồ thị có hướng

Hình 1.1: So sánh giữa đồ thị vô hướng và có hướng

Một đồ thị là vô hướng (undirected) khi cạnh không được chỉ định hướng. Nếu đồ thị tồn tại một cạnh uv, ta có thể đi theo hướng $u \to v$ và hướng $v \to u$. Khi này, việc viết 2 cạnh uv và vu là như nhau và ta chỉ cần viết 1 trong 2 cạnh.

Một đồ thị là có hướng (directed) khi cạnh được chỉ định hướng. Điều này có nghĩa rằng nếu đồ thị tồn tại một cạnh uv, ta chỉ có thể đi theo hướng $u \to v$. Khi này, 2 cạnh uv và vu phân biệt.

1.1.1 Đỉnh kề, cạnh liền thuộc, đỉnh

Xét về đỉnh, hai đỉnh u và v của đơn đồ thị vô hướng $G = (V, E)$ được gọi là kề nhau (adjacent) nếu $uv \in E$, hay nói cách khác, uv là một cạnh của G. Khi đó, ta gọi cạnh uv là cạnh liên thuộc (incident) với hai đỉnh u và v, đồng thời u và v là hai đỉnh

đầu mút (endpoints) của cạnh uv.

Một đỉnh $u \in V$ thì tập hợp các đỉnh hàng xóm (neighbours) của u sẽ bao gồm tất cả các đỉnh $v \in V$ sao cho $uv \in E$, tức là tập hợp tất cả các đỉnh v kề với u. Bậc (degree) của đỉnh u chính là số lượng hàng xóm của đỉnh u. Ký hiệu $\deg(u)$ biểu thị bậc của đỉnh u, tức là số cạnh kề với đỉnh đó trong đồ thị. Cùng với đó theo định lý bắt tay (Handshaking Lemma) trong lý thuyết đồ thị, áp dụng cho đồ thị vô hướng:

$$\sum_{u \in V} \deg(u) = 2|E|$$

Nếu G là một đồ thị có hướng thì bậc ra (out-degree) của đỉnh u, kí hiệu $\deg^+(u)$, là số lượng cạnh xuất phát từ đỉnh u, hay giá trị của $|\{v \in V \mid uv \in E\}|$.

Bậc vào (in-degree) của đỉnh u, kí hiệu $\deg^-(u)$, là số lượng cạnh kết thúc tại đỉnh u, hay giá trị của $|\{v \in V \mid vu \in E\}|$.

Trong đồ thị có hướng, tổng bậc vào của tất cả các đỉnh luôn bằng tổng bậc ra của tất cả các đỉnh (vì mỗi cạnh có một đỉnh bắt đầu và một đỉnh kết thúc).

1.1.2 Đường đi, chu trình

Với $w = (v_0, v_1, v_2, \ldots, v_k)$ là một đường đi trong G, ta có:

- $v_0, v_1, v_2, \ldots, v_k$ là các đỉnh của w.
- $v_0v_1, v_1v_2, \ldots, v_{k-1}v_k$ là các cạnh của w.

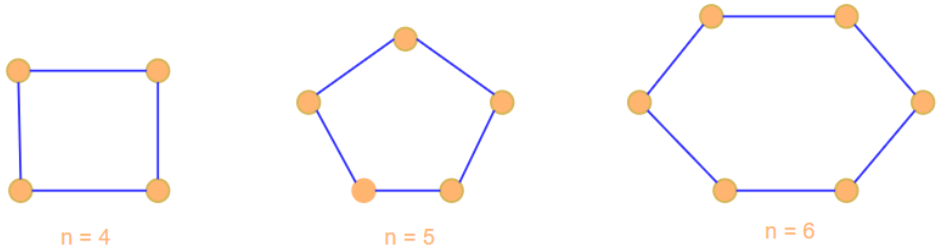
Đường đi (walk) [2] trong đồ thị G là một dãy các đỉnh $(v_0, v_1, v_2, \ldots, v_k)$ thuộc G sao cho tồn tại một dãy các cạnh $(v_0v_1, v_1v_2, \ldots, v_{k-1}v_k)$, trong đó mỗi cặp v_iv_{i+1} là một cạnh của đồ thị. Một đường đi có thể chứa các đỉnh hoặc các cạnh lặp lại.

Trail là một đường đi trong đó tất cả các cạnh trên đường đi đôi một phân biệt [2]. Một path là một đường đi trong đó tất cả các đỉnh trên đường đi đôi một phân biệt (do đó, các cạnh trên đường đi cũng đôi một phân biệt).

Độ dài của một đường đi w trong đồ thị là một số nguyên không âm k, trong đó k là số cạnh xuất hiện trong đường đi w. Khi đó, đường đi sẽ đi qua $k+1$ đỉnh theo thứ tự liên tiếp. Nếu G có trọng số, độ dài của đường đi là tổng trọng số của các cạnh trên đường đi.

- v_0 được gọi là đỉnh đầu (starting point) của w, ta nói w *bắt đầu* tại đỉnh v_0.
- v_k được gọi là đỉnh cuối (ending point) của w, ta nói w *kết thúc* tại đỉnh v_k.

- Cho hai đỉnh p và q thuộc G, ta nói đường đi từ p đến q là đường đi bắt đầu từ đỉnh p và kết thúc tại đỉnh q.



Hình 1.2: Chu trình đồ thị

Một đường đi khép kín [2] (closed walk) của G là một đường đi mà đỉnh cuối trùng với đỉnh đầu. Hay nói cách khác, là một dãy các đỉnh $(v_0, v_1, v_2, \ldots, v_k)$ với $v_0 = v_k$.

Một chu trình của đồ thị G là một đường đi khép kín có dạng $(v_0, v_1, v_2, \ldots, v_k)$ với $k \geq 3$ và các đỉnh $v_0, v_1, v_2, \ldots, v_{k-1}$ đôi một phân biệt. Ngoài ra còn một số trường hợp đặc biệt sau:

(a) Chu trình có 2 đỉnh

(b) Chu trình một đỉnh

Hình 1.3: Các dạng chu trình đặc biệt

- **Chu trình có 2 đỉnh:** Nếu G là một đồ thị có hướng hoặc là một đa đồ thị, thì G có thể tồn tại một chu trình chỉ gồm hai đỉnh khi có hai đỉnh a và b sao cho giữa chúng tồn tại hai cạnh song song (một từ a đến b và một từ b đến a).
- **Chu trình với một đỉnh (self-loop):** Nếu một đồ thị có chứa một cạnh (v, v), tức là một cạnh bắt đầu và kết thúc tại cùng một đỉnh, thì chu trình này chỉ bao gồm một đỉnh duy nhất. Đây được gọi là cạnh khuyên (self-loop).

1.1.3 Tính liên thông, khớp, cầu

Hai đỉnh u và v của đồ thị G được gọi là liên thông nếu tồn tại ít nhất một đường đi từ u đến v tức là 2 đỉnh đó được nối bởi 1 cạnh. Đồ thị G được gọi là liên thông nếu với mọi cặp đỉnh $u, v \in V$, luôn tồn tại một đường đi nối u và v. Điều này có nghĩa là

không có đỉnh nào bị tách rời khỏi phần còn lại của đồ thị.

Trong đồ thị có hướng, tính liên thông được phân loại thành hai loại chính: liên thông mạnh và liên thông yếu. Một đồ thị có hướng $G = (V, A)$ được gọi là liên thông mạnh (strongly connected) nếu với mọi cặp đỉnh $u, v \in V$, luôn tồn tại đường đi có hướng từ u đến v và đường đi có hướng từ v đến u. Nói cách khác, nếu có thể đi từ bất kỳ đỉnh nào đến bất kỳ đỉnh nào khác bằng các cung có hướng trong đồ thị, thì đồ thị đó liên thông mạnh.

Một đồ thị có hướng [2] $G = (V, A)$ được gọi là liên thông yếu (weakly connected) nếu khi ta bỏ hướng của tất cả các cung và chỉ xét đồ thị như một đồ thị vô hướng, thì đồ thị đó liên thông. Nói cách khác, nếu khi bỏ hướng của các cung, ta có một đồ thị vô hướng liên thông, thì đồ thị gốc được gọi là liên thông yếu.

(a) Liên thông mạnh (b) Liên thông yếu

Hình 1.4: So sánh đồ thị liên thông mạnh và liên thông yếu

Mọi đồ thị liên thông mạnh đều liên thông yếu, nhưng không phải mọi đồ thị liên thông yếu đều liên thông mạnh. Nếu một đồ thị liên thông mạnh, nghĩa là mọi cặp đỉnh đều có đường đi hai chiều giữa chúng, nên khi bỏ hướng, đồ thị vẫn sẽ liên thông. Tuy nhiên, một đồ thị có thể liên thông yếu nhưng không liên thông mạnh nếu có một số cặp đỉnh không có đường đi hai chiều giữa chúng.

Nếu G không liên thông, đồ thị G sẽ bao gồm nhiều đồ thị con liên thông, rời nhau. Mỗi đồ thị con như vậy được gọi là một thành phần liên thông (TPLT) (connected component) của G.

Một đỉnh u thuộc đồ thị vô hướng $G = (V, E)$ được gọi là một khớp (articulation point) nếu khi ta xóa đỉnh u khỏi đồ thị (cùng với tất cả các cạnh liên thuộc với nó), thì đồ thị không còn liên thông. Nói cách khác, nếu việc loại bỏ u làm tăng số thành phần liên thông của G, thì u là một khớp.

Một cạnh $e = (u, v)$ thuộc đồ thị vô hướng G được gọi là một cầu (bridge) nếu

khi ta xóa cạnh uv, đồ thị không còn liên thông. Nói cách khác, nếu loại bỏ uv làm tăng số thành phần liên thông của G, thì uv là một cầu.

1.1.4 Cây, gốc, lá và quan hệ giữa các đỉnh trong cây

Cho một đồ thị $G = (V, E)$, ta có các định nghĩa sau: Một đồ thị G được gọi là **cây** nếu nó thỏa mãn ít nhất hai trong ba điều kiện dưới đây:

- G không chứa chu trình (acyclic).
- G liên thông (connected).
- Số cạnh của G bằng số đỉnh trừ đi 1, tức là $|E(G)| = |V(G)| - 1$.

Khi G [2] là một cây thì sẽ chỉ tồn tại một đường đi đơn duy nhất giữa hai đỉnh bất kỳ. Nếu ta thêm một cạnh bất kỳ vào G mà chưa có sẵn trong G, thì sẽ xuất hiện một chu trình. Nếu ta xóa một cạnh bất kỳ khỏi G, số thành phần liên thông của G sẽ tăng lên, do đó mọi cạnh trong cây đều là cạnh cầu.

Một đồ thị G được gọi là rừng cây (forest) nếu nó có nhiều hơn một thành phần liên thông (TPLT), và mỗi TPLT là một cây. Một cây (tree) là một đồ thị liên thông không có chu trình. Một đồ thị cây có thể có hoặc không có gốc, tùy vào cách định nghĩa.

Cho một đồ thị cây $T = (V, E)$, trong đó V là tập hợp các đỉnh và E là tập hợp các cạnh của cây, ta có các khái niệm quan trọng sau:

1. **Gốc của cây (root)**
   Một cây được xác định cùng với một đỉnh đặc biệt được chọn làm gốc, gọi là đỉnh gốc (root). Gốc là điểm xuất phát để xác định quan hệ cha – con giữa các đỉnh trong cây. Nếu không có quy định cụ thể, ta có thể chọn một đỉnh bất kỳ trong tập đỉnh V để làm gốc, chẳng hạn chọn đỉnh có chỉ số nhỏ nhất trong V để thống nhất cách xử lý.
2. **Đỉnh lá (leaf)**
   Một đỉnh lá là một đỉnh có bậc đúng bằng 1, tức là nó chỉ liên kết với duy nhất một đỉnh khác trong cây. Trong cấu trúc cây, các đỉnh lá là những đỉnh không tiếp tục phân nhánh thêm và thường là điểm kết thúc của các nhánh.
3. **Quan hệ cha-con (parent-child)**
   Nếu cây có gốc, các đỉnh trong cây sẽ có mối quan hệ cha-con, cụ thể như sau:
   - Xét một cạnh uv, nếu v gần gốc hơn u, thì v là cha (parent) của u, và u là con

(child) của v.

- Một đỉnh có thể có nhiều con, nhưng chỉ có duy nhất một cha.
- Đỉnh gốc không có cha, trong khi các đỉnh lá không có con.

4. **Quan hệ tổ tiên - hậu duệ (ancestor-descendant)**
   - Một đỉnh u được gọi là tổ tiên (ancestor) của một đỉnh v nếu trên đường từ v đến gốc có đỉnh u.
   - v được gọi là hậu duệ (descendant) của u nếu u là tổ tiên của v.
   - Trong một cây có gốc, *tổ tiên cấp* k (k-th ancestor) của một đỉnh v là đỉnh u nằm trên đường đi duy nhất từ v tới gốc sao cho độ dài đường đi (số cạnh) từ v đến u là đúng k.
5. **Chiều cao và chiều sâu của đỉnh trong cây**
   - Chiều cao (height) của một đỉnh v là khoảng cách từ v đỉnh con cháu của v xa nhất trong cây.
   - Chiều sâu (depth) của một đỉnh v là khoảng cách từ v đến gốc.
   - Chiều cao của cây T là giá trị lớn nhất trong tất cả các chiều cao của các đỉnh trong cây.
6. **Đường kính của cây (diameter)**

   Đường kính của cây là khoảng cách dài nhất giữa hai đỉnh trong cây. Có thể tìm đường kính bằng cách:

   (a) Chọn một đỉnh bất kỳ u.

   (b) Tìm đỉnh v xa nhất so với u.

   (c) Tìm đỉnh w xa nhất so với v, khi đó $d(v, w)$ chính là đường kính của cây.

1.2 Một số lớp bài toán khó trong lý thuyết độ phức tạp

1.2.1 PSPACE-complete

Trong lý thuyết độ phức tạp tính toán, PSPACE (Polynomial Space) là một lớp bài toán quan trọng, tập trung vào bộ nhớ sử dụng thay vì thời gian chạy. Một bài toán thuộc PSPACE nếu nó có thể được giải bằng một thuật toán sử dụng bộ nhớ đa thức theo kích thước đầu vào. Điều này có nghĩa là mặc dù thời gian chạy có thể rất lớn, miễn là lượng bộ nhớ sử dụng không vượt quá một hàm đa thức của đầu vào, bài toán đó vẫn nằm trong PSPACE.

Một cách hình thức, PSPACE được định nghĩa như sau:

$$\text{PSPACE} = \bigcup_{k \geq 1} \text{SPACE}(n^k)$$

Tức là, PSPACE bao gồm tất cả các bài toán có thể được giải bởi một máy Turing với bộ nhớ bị giới hạn bởi một hàm đa thức theo kích thước đầu vào.

Không gian tính toán SPACE($f(n)$) của một thuật toán được định nghĩa là số ô nhớ tối đa mà thuật toán sử dụng trong quá trình thực thi. Một số cấp độ không gian phổ biến bao gồm:

- **Không gian đa thức**: Thuật toán sử dụng không quá $O(n^k)$ ô nhớ, với k là một hằng số [3].
- **Không gian lũy thừa** (EXP): Sử dụng tối đa $O(2^{n^k})$ ô nhớ [4]

Một bài toán được gọi là PSPACE-complete nếu thỏa mãn hai điều kiện sau:

- Nó thuộc PSPACE, tức là có thể giải được bằng một thuật toán sử dụng bộ nhớ đa thức theo kích thước đầu vào [5]
- Nó là PSPACE-hard, tức là mọi bài toán trong lớp PSPACE đều có thể được giảm về nó trong thời gian đa thức. [6].

Để chứng minh một bài toán thuộc PSPACE-complete, ta thường sử dụng kỹ thuật giảm trong thời gian đa thức. Khi chứng minh một bài toán B là PSPACE-complete, ta thường thực hiện bằng cách chọn một bài toán A đã biết thuộc PSPACE-complete và xây dựng một phép giảm từ A về B sao cho bất kỳ lời giải nào của A cũng có thể được ánh xạ sang một lời giải hợp lệ của B. Nếu phép giảm này có thể được thực hiện trong thời gian đa thức, điều đó chứng tỏ rằng B cũng khó ít nhất như mọi bài toán trong PSPACE, và do đó, B thuộc PSPACE-complete.

Các bài toán PSPACE-complete được xem là những bài toán khó nhất trong lớp PSPACE. Nếu có một thuật toán thời gian đa thức giải được một bài toán PSPACE-complete, thì mọi bài toán trong PSPACE cũng có thể giải quyết trong thời gian đa thức, tức là:

$$P = PSPACE.$$

Do đó, nếu có một thuật toán chạy trong thời gian $O(n^k)$ cho một bài toán PSPACE-complete, thì nó cũng có thể giải mọi bài toán trong PSPACE với cùng giới hạn thời gian, dẫn đến một kết quả quan trọng trong lý thuyết độ phức tạp.

1.2.2 NP-complete

Trong lý thuyết độ phức tạp tính toán, NP (Nondeterministic Polynomial Time) là tập hợp các bài toán quyết định mà nghiệm của chúng có thể được kiểm tra trong thời gian đa thức. NP là một trong những lớp độ phức tạp quan trọng nhất, đóng vai trò nền tảng trong nghiên cứu về thuật toán, tối ưu hóa và mật mã.

Một bài toán quyết định L thuộc NP nếu tồn tại một máy Turing không xác định (Nondeterministic Turing Machine - NTM) có thể giải quyết bài toán trong thời gian đa thức sao cho:

- Với mọi đầu vào $x \in L$, tồn tại một chuỗi các lựa chọn để M chấp nhận x trong thời gian đa thức.
- Nếu $x \notin L$, mọi chuỗi lựa chọn đều dẫn đến việc từ chối x.

Khi đó, ta có:

$$\text{NP} = \bigcup_{k \geq 1} \text{NTIME}(n^k)$$

Trong đó, NTIME(n^k) là tập hợp các bài toán có thể giải bởi máy Turing không xác định trong thời gian $O(n^k)$.

NP-complete (NP-đầy đủ) là tập hợp các bài toán quyết định khó nhất trong lớp NP (Nondeterministic Polynomial Time). Nếu một bài toán NP-complete có thể được giải trong thời gian đa thức, thì mọi bài toán trong NP cũng có thể giải trong thời gian đa thức, tức là P = NP [7] .

Một bài toán L thuộc NP-complete nếu thỏa mãn hai điều kiện sau:

- $L \in NP$ (tức là có thể kiểm tra nghiệm trong thời gian đa thức).
- L là NP-hard, nghĩa là mọi bài toán trong NP có thể giảm về L trong thời gian đa thức:

$$\forall L' \in NP, \quad L' \leq_p L.$$

Trong đó $L' \leq_p L$ có nghĩa là tồn tại một thuật toán giảm bài toán L' về bài toán L trong thời gian đa thức. Điều này có nghĩa là nếu tồn tại một thuật toán thời gian đa thức giải quyết một bài toán NP-complete, thì mọi bài toán trong NP cũng có thể được giải trong thời gian đa thức.

Nếu có một thuật toán giải quyết một bài toán NP-complete trong thời gian đa

thức, thì ta suy ra rằng:

$$P = NP$$

Tuy nhiên, hiện nay chưa có thuật toán nào giải một bài toán NP-complete trong thời gian đa thức, và câu hỏi $P \stackrel{?}{=} NP$ vẫn là một trong những vấn đề chưa có lời giải quan trọng nhất trong khoa học máy tính.

1.3 Các ký hiệu quan trọng

Trong lý thuyết đồ thị, việc sử dụng các ký hiệu chuẩn hóa giúp mô tả chính xác các khái niệm và thuật toán liên quan đến đồ thị có hướng. Mục tiêu của phần này là thiết lập các định nghĩa và ký hiệu quan trọng để hỗ trợ các phân tích và chứng minh sau này. Các ký hiệu và định nghĩa trong phần này sẽ được sử dụng xuyên suốt trong báo cáo khóa luận nhằm đảm bảo tính nhất quán khi phân tích bài toán trên đồ thị có hướng.

1.3.1 Cấu trúc đồ thị

Cho $G = (V, A)$ là một đồ thị có hướng. Chúng ta sẽ ký hiệu tập đỉnh và tập cung của G lần lượt là $V(G)$ và $A(G)$. Xét một cung $e = (u, v) \in A(G)$, trong đó:

- Đỉnh u được gọi là gốc (tail) của cung e, tức là cung e xuất phát từ đỉnh u.
- Đỉnh v được gọi là đầu (head) của cung e, tức là cung e kết thúc tại đỉnh v.
- Hai đỉnh u và v được gọi là đỉnh đầu mút (endpoints) của cung e.

Nói cách khác, cung $e = (u, v)$ biểu diễn một mối quan hệ có hướng từ đỉnh u đến đỉnh v, trong đó u là điểm bắt đầu của cung và v là điểm kết thúc.

Với một đồ thị có hướng G, đồ thị vô hướng nền (underlying undirected graph) của G là một đồ thị vô hướng được tạo ra bằng cách bỏ qua hướng của tất cả các cung trong G. Nói cách khác, nếu $G = (V, A)$ là một đồ thị có hướng, thì đồ thị vô hướng nền G^{und} có cùng tập đỉnh V nhưng tập cạnh được xác định như sau:

$$E(G^{\text{und}}) = \{\{u, v\} \mid (u, v) \in A(G) \text{ hoặc } (v, u) \in A(G)\}.$$

Tức là, nếu có một cung từ u đến v hoặc từ v đến u trong G, thì trong G^{und}, ta có một cạnh vô hướng giữa u và v. Chúng ta ký hiệu đồ thị vô hướng nền của G là G^{und}.

1.3.2 Tập độc lập trong đồ thị có hướng

Cho một đồ thị vô hướng $G' = (V, E)$, trong đó:

- $V(G')$ là tập hợp tất cả các đỉnh của đồ thị G'.
- $E(G')$ là tập hợp tất cả các cạnh nối giữa các đỉnh trong $V(G')$, tức là $E(G') \subseteq \{\{u, v\} \mid u, v \in V(G'), u \neq v\}$.

Một tập con $S \subseteq V(G')$ được gọi là tập độc lập (Independent Set) nếu không có hai đỉnh nào trong S được nối với nhau bởi một cạnh trong $E(G')$. Điều kiện để một tập S là tập độc lập có thể được viết lại dưới dạng:

$$\forall u, v \in S, \quad u \neq v \Rightarrow \{u, v\} \notin E(G')$$

Tức là, với mọi cặp đỉnh phân biệt u, v thuộc tập S, không có cạnh nào nối giữa chúng trong đồ thị. Tập S là tập độc lập trong đồ thị có hướng G nếu với mọi cặp đỉnh phân biệt $u, v \in S$, không tồn tại cung nào nối giữa u và v trong G, tức là:

$$\forall u, v \in S, \quad \{u, v\} \notin E(G^{\text{und}})$$

Nghĩa là tập S là tập độc lập trong G nếu khi ta bỏ qua hướng của các cung trong G, không có cạnh nào trong đồ thị vô hướng nền G^{und} nối giữa hai đỉnh bất kỳ trong S.

Định nghĩa này đảm bảo rằng khi xét một tập con của đỉnh trong một đồ thị có hướng, ta có thể kiểm tra tính độc lập của nó bằng cách quy chiếu về đồ thị vô hướng tương ứng. Điều này có nghĩa là hướng của cung không ảnh hưởng đến tính độc lập của tập đỉnh, mà chỉ cần kiểm tra mối quan hệ tồn tại giữa các đỉnh khi bỏ qua hướng.

1.3.3 Quan hệ giữa các đỉnh và cạnh

Cho đồ thị có hướng $G = (V, A)$ và hai đỉnh $u, v \in V(G)$. Ta nói rằng hai đỉnh u và v là kề nhau nếu chúng cũng kề nhau trong đồ thị vô hướng nền G^{und} của G. Hai đỉnh u và v được gọi là kề nhau (adjacent) trong G nếu:

$$\{u, v\} \in E(G^{\text{und}})$$

tức là, tồn tại ít nhất một cung có hướng từ u đến v hoặc từ v đến u trong G.

Trong đồ thị có hướng, hai đỉnh có thể có quan hệ có hướng (tức là có cung từ u đến v nhưng không có cung ngược lại). Khi xét trong G^{und}, hướng của các cung không còn quan trọng, ta chỉ quan tâm đến việc liệu có tồn tại kết nối giữa hai đỉnh

hay không. Như vậy, nếu u và v có ít nhất một cung nối chúng trong G, chúng sẽ được xem là kề nhau trong G^{und}, và do đó cũng được gọi là kề nhau trong G.

Cho một cung $e \in A(G)$ và một đỉnh $v \in V(G)$, ta nói rằng cung e được gọi là gắn với (incident to) đỉnh v nếu v là một trong hai đỉnh đầu mút của e. Điều này có nghĩa là v có thể là:

- Đỉnh đầu của cung nếu $e = (u, v)$, tức là có một cung đi từ u đến v.
- Đỉnh gốc của cung nếu $e = (v, w)$, tức là có một cung đi từ v đến w.

Xét một cung $(u, v) \in A(G)$. Khi đó, đỉnh u được gọi là láng giềng vào (in-neighbor) của v, vì có một cung đi từ u đến v. Đỉnh v được gọi là láng giềng ra (out-neighbor) của u, vì có một cung xuất phát từ u đến v. Nói chung, hai đỉnh u và v được gọi là láng giềng (neighbor) của nhau nếu tồn tại ít nhất một cung nối giữa chúng, bất kể hướng của cung. Với một đỉnh v chúng ta ký hiệu như sau:

- Tập các láng giềng ra (out-neighbors) của v tức là tập hợp các đỉnh mà có cung đi từ v đến chúng, ký hiệu là $N_G^+(v)$:
$$N_G^+(v) := \{u \in V(G) \mid (v, u) \in A(G)\}$$
- Tập các láng giềng vào (in-neighbors) của v, tức là tập hợp các đỉnh có cung đi vào v ký hiệu là $N_G^-(v)$:
$$N_G^-(v) := \{u \in V(G) \mid (u, v) \in A(G)\}$$
- Tập láng giềng (neighbors) của v bao gồm cả láng giềng vào và láng giềng ra được ký hiệu là $N_G(v)$:
$$N_G(v) := N_G^+(v) \cup N_G^-(v)$$
- Tập láng giềng mở rộng (closed neighborhood) của v bao gồm tất cả các đỉnh kề với v, cùng với chính v được ký hiệu là $N_G[v]$:
$$N_G[v] := N_G(v) \cup \{v\}$$
- Tập các cung đi ra (outgoing arcs) từ v, chứa tất cả các cung có gốc là v và đi đến một đỉnh khác trong đồ thị ký hiệu là $\Gamma_G^+(v)$:
$$\Gamma_G^+(v) := \{(v, u) \mid u \in N_G^+(v)\}$$
- Tập các cung đi vào (incoming arcs) của v, chứa tất cả các cung có đỉnh đầu là v và có gốc từ một đỉnh khác trong đồ thị được ký hiệu là $\Gamma_G^-(v)$:
$$\Gamma_G^-(v) := \{(u, v) \mid u \in N_G^-(v)\}$$

- Tập tất cả các cung gắn với (incident arcs) v, chứa tất cả các cung có liên quan đến v, bao gồm cả cung đi vào và cung đi ra được ký hiệu là $\Gamma_G(v)$:

$$\Gamma_G(v) := \Gamma_G^+(v) \cup \Gamma_G^-(v)$$

Nếu ngữ cảnh rõ ràng và không gây nhầm lẫn, ta có thể bỏ chỉ số G trong các ký hiệu trên, viết đơn giản hơn như sau:

$$\Gamma^+(v), \quad \Gamma^-(v), \quad \Gamma(v)$$

Điều này giúp thuận tiện hơn trong các chứng minh và biểu diễn thuật toán.

1.3.4 Ghép các đường đi có hướng

Một đường đi có hướng (Directed Path) P là một dãy các đỉnh và các cung của đồ thị có hướng $G = (V, A)$, được biểu diễn dưới dạng:

$$P = (v_1, e_1, v_2, e_2, \ldots, e_{\ell-1}, v_\ell)$$

sao cho tất cả các đỉnh $v_1, v_2, \ldots, v_\ell$ đều phân biệt với mọi $i \in [\ell - 1]$, cung e_i thỏa mãn:

$$e_i = (v_i, v_{i+1}) \in A(G)$$

tức là, e_i là một cung đi từ v_i đến v_{i+1}. Nguồn (source) của đường đi P là v_1, đích (sink) của đường đi P là v_ℓ, được ký hiệu lần lượt là :

$$s(P) = v_1, \quad t(P) = v_\ell$$

Khi đó, P được gọi là đường đi có hướng từ v_1 đến v_ℓ, hay còn gọi là đường đi v_1-v_ℓ (v_1-v_ℓ path). Độ dài của đường đi P, ký hiệu là $|P|$, được định nghĩa là số lượng cung trong P, tức là:

$$|P| = \ell - 1$$

Với một đường đi có hướng P có độ dài ít nhất 2 ($|P| \geq 2$), ta định nghĩa:

- Đỉnh thứ hai của đường đi, ký hiệu là $s'(P)$, với $s'(P) := v_2$.
- Đỉnh kế cuối của đường đi, ký hiệu là $t'(P)$, là $t'(P) := v_{\ell-1}$.

Với hai tập đỉnh X và Y có cùng kích thước k trên một đồ thị có hướng G, một tập hợp các đường đi có hướng:

$$P = \{P_1, P_2, \ldots, P_k\}$$

được gọi là một phép ghép các đường đi có hướng (Directed Path Matching - DPM) từ X đến Y nếu thỏa mãn các điều kiện sau:

- Các nguồn của các đường đi là các phần tử phân biệt trong X nghĩa là, mỗi đỉnh trong X là nguồn của đúng một đường đi trong tập hợp P:

$$\{s(P_i) \mid i \in [k]\} = X$$

- Các đích của các đường đi là các phần tử phân biệt trong Y, nghĩa là, mỗi đỉnh trong Y là đích của đúng một đường đi trong tập hợp P:

$$\{t(P_i) \mid i \in [k]\} = Y$$

Lưu ý: Hai tập X và Y có thể giao nhau trong định nghĩa này.

Chương 2

BỐI CẢNH NGHIÊN CỨU VÀ CÁC KẾT QUẢ LIÊN QUAN

2.1 Bối cảnh bài toán và các khái niệm tái cấu hình

Một bài toán tái cấu hình (reconfiguration problem) được định nghĩa như sau: ta được cho một bài toán quyết định (decision problem) cùng với hai nghiệm hợp lệ I^s và I^t. Mục tiêu là kiểm tra xem liệu có thể biến đổi dần nghiệm I^s thành I^t thông qua một loạt các bước thay đổi nhỏ (modification rule), sao cho mọi trạng thái trung gian trong quá trình đều là nghiệm hợp lệ của bài toán ban đầu.

Mô hình tái cấu hình không chỉ đặt ra bài toán quyết định truyền thống (có nghiệm hay không), mà còn cung cấp cách tiếp cận động, mô phỏng quá trình chuyển đổi giữa các cấu hình khả thi. Điều này rất hữu ích trong các hệ thống cần thay đổi cấu trúc một cách liên tục và an toàn, chẳng hạn như tối ưu hóa mạng, robot học, lập kế hoạch hành động, hay bảo trì hệ thống đang hoạt động.

Các bài toán tái cấu hình đã được nghiên cứu cho nhiều lớp bài toán tổ hợp quen thuộc, bao gồm:

- Independent Set Reconfiguration (Tái cấu hình tập độc lập) – [8–18]
- Dominating Set Reconfiguration - [19–21]
- Clique Reconfiguration - [22]
- Matching Reconfiguration - [23–25]
- Graph Coloring Reconfiguration - [26–28]

Các hướng tiếp cận và kết quả chính trong lĩnh vực này đã được tổng hợp trong một số bài khảo sát chuyên sâu như [29] và [30]. Trong số đó, tái cấu hình tập độc lập là một trong những bài toán được nghiên cứu rộng rãi nhất, với nhiều biến thể của quy tắc thay đổi và độ phức tạp khác nhau tùy thuộc vào cấu trúc đồ thị và loại thao tác được phép thực hiện. Những nghiên cứu này đặt nền móng quan trọng cho việc mở rộng bài

toán sang các biến thể phức tạp hơn như Directed Token Sliding (viết tắt là DTS), vốn là trọng tâm của khóa luận này. Nhiều nghiên cứu trước đây đã đề xuất và phân tích ba quy tắc biến đổi chính được sử dụng trong quá trình tái cấu hình tập độc lập:

1. Token Addition and Removal (TAR) - [9, 31]
   Quy tắc này cho phép thêm hoặc xóa một đỉnh khỏi tập độc lập hiện tại trong mỗi bước, với điều kiện kích thước của tập vẫn luôn lớn hơn hoặc bằng một ngưỡng tối thiểu được cho trước. Quy tắc này linh hoạt và thường được sử dụng trong các mô hình có yếu tố nới lỏng về số lượng phần tử trong tập tại mỗi bước.
2. Token Jumping (TJ) - [15, 32, 33]
   Ở quy tắc này, ta có thể di chuyển một token từ một đỉnh trong tập độc lập hiện tại đến bất kỳ đỉnh nào chưa nằm trong tập, không yêu cầu hai đỉnh phải kề nhau. Điều này cho phép nhảy qua khoảng cách lớn nhưng vẫn đảm bảo tính chất độc lập được duy trì trong suốt quá trình.
3. Token Sliding (TS) - [8, 10–18]
   Đây là phiên bản giới hạn hơn của Token Jumping, trong đó token chỉ được phép di chuyển giữa hai đỉnh kề nhau trong đồ thị. Điều này mô phỏng chính xác hơn các mô hình vật lý, nơi một đối tượng không thể "biến mất" ở một vị trí và "xuất hiện" ở nơi khác mà không đi qua các bước trung gian.

Trong nghiên cứu, khi nói đến tái cấu hình tập độc lập theo quy tắc Token Sliding, người ta thường gọi tắt là TS.

Một điểm đáng lưu ý là cả ba quy tắc kể trên đều mang tính đối xứng (symmetric). Cụ thể, nếu ta có thể biến đổi tập độc lập I^s thành tập I^t thông qua một t chuỗi các bước hợp lệ, thì cũng luôn có thể thực hiện quá trình ngược lại, từ I^t về I^s, bằng cách áp dụng các bước ngược theo đúng quy tắc tương ứng. Tuy nhiên, điều gì sẽ xảy ra nếu ta áp dụng một quy tắc không đối xứng (asymmetric) – chẳng hạn như chỉ cho phép trượt token theo một chiều nhất định trong đồ thị có hướng? Câu hỏi này mở ra một hướng nghiên cứu mới, chính là trọng tâm của bài toán Directed Token Sliding mà khóa luận này đề cập.

2.2 Các nghiên cứu liên quan trên đồ thị vô hướng

Xét trên trường hợp Token Sliding trên đồ thị vô hướng, bài toán tái cấu hình đã được nghiên cứu kỹ lưỡng trước đó. Đặc biệt, trên đồ thị là cây (undirected trees),

Demaine và cộng sự [11] đã xây dựng một thuật toán tuyến tính để kiểm tra khả năng tái cấu hình giữa hai tập độc lập I^s và I^t. Thuật toán này cho phép xác định một cách hiệu quả liệu có tồn tại chuỗi các bước trượt token hợp lệ từ I^s đến I^t hay không.

Ngoài ra, cũng trong nghiên cứu của Demaine et al. [11], nhóm tác giả đề xuất thuật toán có độ phức tạp $O(|V|^2)$ để xây dựng một chuỗi tái cấu hình (nếu tồn tại). Tuy nhiên, chuỗi được tạo ra không được đảm bảo là ngắn nhất (shortest reconfiguration sequence). Để giải quyết bài toán tìm chuỗi tái cấu hình ngắn nhất trên cây vô hướng, nghiên cứu sau đó của Ito và cộng sự [34] đã đề xuất thuật toán có độ phức tạp cao hơn lên tới $O(|V|^5)$. Đây hiện là kết quả tốt nhất được biết với mục tiêu tối ưu độ dài chuỗi.

Ngược lại, với bài toán Directed Token Sliding trên đồ thị có hướng, kết quả chính trong bài báo gốc cho thấy rằng: có thể xây dựng một chuỗi tái cấu hình ngắn nhất trên đồ thị polytree trong thời gian $O(|V|^2)$ [1] và đây là kết quả tối ưu (optimal) vì đã đạt tới cận dưới cho một số trường hợp (ví dụ như đường đi có hướng – directed path). Tuy nhiên, nhóm tác giả cũng lưu ý rằng thuật toán bậc hai $O(|V|^2))$ [1] này không thể được chuyển trực tiếp sang trường hợp vô hướng, bởi vì polytree không cho phép tồn tại các cung song song ngược chiều (antiparallel arcs). Điều này thể hiện bản chất không đối xứng trong bài toán tái cấu hình trên đồ thị có hướng.

Bài toán Token Sliding trong đồ thị vô hướng lần đầu tiên được giới thiệu bởi Hearn và Demaine [8]. Trong nghiên cứu này, hai tác giả chỉ ra rằng bài toán có độ phức tạp PSPACE-complete, ngay cả khi bị giới hạn trên đồ thị phẳng (planar graphs) với bậc tối đa là 3. Điều này cho thấy tính chất tính toán rất khó của bài toán, ngay cả trên những lớp đồ thị có cấu trúc đơn giản. Ngoài ra, bài toán Token Sliding cũng được chứng minh là PSPACE-complete trên các lớp đồ thị đặc biệt khác, chẳng hạn như đồ thị hai phía (bipartite graphs) [16] và đồ thị tách (split graphs) [17].

Tuy nhiên, trong số các kết quả tiêu cực nêu trên, vẫn tồn tại một số lớp đồ thị mà bài toán có thể giải được trong thời gian đa thức. Cụ thể, bài toán Token Sliding được chứng minh là giải được trong thời gian đa thức trên các lớp đồ thị như:

- Cographs – lớp đồ thị không chứa đường đi độ dài 4 làm đồ thị con cảm ứng [9].
- Claw-free graphs – đồ thị không chứa cây hình móng vuốt $K_{1,3}$ làm đồ thị con cảm ứng [10].
- Cây (trees) – đồ thị vô hướng không chu trình [11].

- Cactus graphs – đồ thị mà mọi cạnh nằm trong nhiều nhất một chu trình [13].
- Interval graphs – đồ thị biểu diễn bằng tập các đoạn trên trục số [14] .
- Permutation graphs, và Bipartite distance-hereditary graphs – lớp con của đồ thị hai phía mà khoảng cách giữa các đỉnh được bảo toàn khi lấy đồ thị con cảm ứng [12].

Sự đa dạng về kết quả cho thấy rằng tính chất cấu trúc của đồ thị đầu vào đóng vai trò quan trọng trong việc quyết định độ phức tạp tính toán của bài toán tái cấu hình.

2.3 Khởi đầu nghiên cứu trên đồ thị có hướng

Trong khi bài toán Token Sliding trên đồ thị vô hướng đã được nghiên cứu sâu rộng và đạt được nhiều kết quả quan trọng, thì trước đây chưa có nghiên cứu chính thức nào đề cập đến bài toán này trên đồ thị có hướng. Việc mở rộng mô hình từ đồ thị vô hướng sang đồ thị có hướng không chỉ là sự thay đổi về mặt cấu trúc, mà còn dẫn đến những khác biệt cơ bản về tính chất bài toán và độ phức tạp thuật toán.

Trước khi bài toán Directed Token Sliding (DTS) được giới thiệu, phần lớn các công trình nghiên cứu về tái cấu hình tập độc lập đều tập trung trong bối cảnh đồ thị vô hướng, nơi các thao tác trượt token có tính đối xứng. Trong bối cảnh này, nhiều kết quả nổi bật đã được thiết lập, đặc biệt là:

- Thuật toán kiểm tra tính tái cấu hình trên cây trong thời gian tuyến tính.
- Thuật toán xây dựng chuỗi tái cấu hình với độ dài $O(|(V|^2)$ [11].

Tuy nhiên, khi chuyển sang đồ thị có hướng, bài toán trở nên bất đối xứng do token chỉ có thể di chuyển theo chiều cung, điều này đặt ra những thách thức mới về mặt lý thuyết và thuật toán – và chính là lý do cần thiết để nghiên cứu một hướng tiếp cận hoàn toàn mới như bài toán Directed Token Sliding.

Mặc dù Token Sliding trước đây chủ yếu được nghiên cứu trên đồ thị vô hướng, và chưa từng được nghiên cứu một cách bài bản trên đồ thị có hướng (digraphs), song vẫn tồn tại một số nhánh nghiên cứu liên quan đến tái cấu hình trên đồ thị có hướng, cụ thể là trong ngữ cảnh định hướng (orientation) và các tiểu đồ thị có hướng. Một số công trình đáng chú ý bao gồm:

- Tái cấu hình orientation mạnh (strong orientations): tức là thay đổi cách định hướng các cạnh trong một đồ thị sao cho kết quả là một đồ thị có hướng mạnh liên thông [35–37].

- Tái cấu hình orientation acyclic (định hướng không chu trình) [36].
- Tái cấu hình trong logic ràng buộc bất định (nondeterministic constraint logic) [8], một mô hình tổng quát hóa nhiều bài toán tái cấu hình.
- Tái cấu hình -orientations, tức là thay đổi orientation sao cho mỗi đỉnh có số lượng cung ra cụ thể theo yêu cầu [38].

Đặc biệt, Ito et al. (2021) [39] gần đây đã mở rộng phạm vi tái cấu hình sang một số tiểu đồ thị có hướng, như cây có hướng (directed trees), đường đi có hướng (directed paths) Và các đồ thị có cấu trúc DAG (directed acyclic subgraphs).

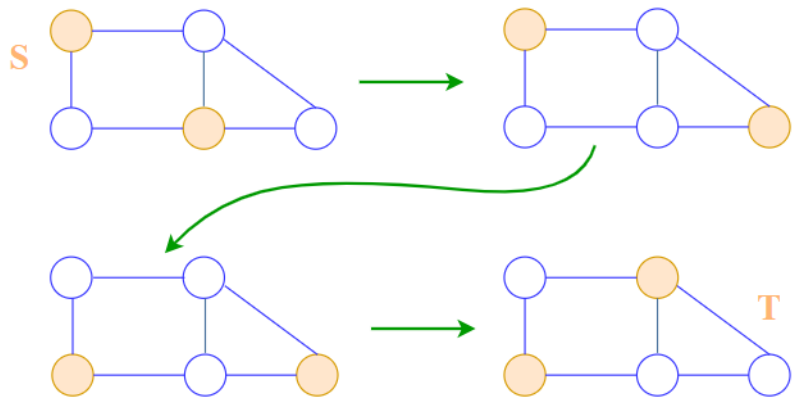
Tuy nhiên, một điểm quan trọng là các quy tắc tái cấu hình trong các nghiên cứu kể trên đều mang tính đối xứng. Nghĩa là, nếu một lời giải X có thể tái cấu hình thành lời giải Y bằng một loạt thao tác, thì Y cũng có thể được tái cấu hình trở lại thành X bằng chính các thao tác đó theo chiều ngược lại. Trong khi đó, bài toán Directed Token Sliding được đề xuất trong nghiên cứu này lại mang bản chất bất đối xứng – token chỉ có thể trượt theo chiều cung, khiến cho việc tái cấu hình không thể quay ngược trở lại một cách tùy ý. Đây chính là điểm mấu chốt tạo nên sự khác biệt trong lý thuyết và độ phức tạp giải thuật của bài toán này so với các mô hình đã có.

2.4 Phát biểu tổng quan về bài toán Directed Token Sliding

Sau khi đã có nhiều kết quả quan trọng trên đồ thị vô hướng, các tác giả trong [Wasa et al., 2022] [1] đã mở rộng bài toán sang trường hợp đồ thị có hướng, và giới thiệu một biến thể mới mang tên Directed Token Sliding (DTS). Trong biến thể này, thay vì cho phép token trượt theo cả hai chiều của cạnh như trong đồ thị vô hướng, mỗi token chỉ được phép trượt dọc theo cung của đồ thị có hướng, tức là chỉ theo đúng chiều của cung đã cho.

Giả sử trong bài toán tái cấu hình có hai nghiệm khả thi I^s và I^t (nghiệm khả thi là nghiệm thỏa mãn tất cả ràng buộc của bài toán). Nhiệm vụ của chúng ta sẽ là biến đổi I^s -> I^t bằng cách lặp lại một quy tắc sửa đổi trong khi vẫn duy trì tính khả thi của nghiệm trung gian. Thông thường sẽ có 3 quy tắc sửa đổi khác nhau như chúng ta đã giới thiệu ở trên đó là Token Addition and Removal (TAR), Token Jumping (TJ) Token Sliding (TS). Ở đây chúng ta sẽ tập trung nghiên cứu tái cấu hình độc lập theo quy tắc Token Sliding hay còn gọi là trượt token. Tất cả các quy tắc trên đều đối xứng tức là các quy tắc này đang được áp dụng trên đồ thị vô hướng.

Trong ví dụ minh họa trong **Hình 2.1**, ta thấy rằng quy tắc TS đang được xem xét trên một đồ thị vô hướng. Ta định nghĩa rằng token là một vật thể hoặc một điểm đánh dấu được đặt trên một số đỉnh trong đồ thị (tương tự như quân cờ trên các ô cờ). Mỗi lần chuyển trạng thái chỉ có một token được di chuyển. Token chỉ có thể di chuyển đến một đỉnh kề cạnh với nó và ở đỉnh đó đang trống. Một điều rất quan trọng đó là sau mỗi lần di chuyển thì tập hợp các đỉnh chứa token vẫn phải là tập hợp độc lập nghĩa là sẽ không có 2 token nào nằm trên 2 đỉnh kề cạnh.



Hình 2.1: Token Sliding trên đồ thị vô hướng

Câu hỏi được đặt ra điều gì sẽ xảy ra nếu chúng ta áp dụng một quy tắc bất đối xứng. Câu trả lời đó là bài toán sẽ được mở rộng sang đồ thị có hướng, độ phức tạp và bản chất của bài toán sẽ thay đổi đáng kể khi mà giờ đây các cạnh của đồ thị sẽ bị ràng buộc về hướng nghĩa là token có thể trượt từ đỉnh này sang đỉnh kia nhưng không thể trượt theo hướng ngược lại. Điều này làm thay đổi hoàn toàn cách thức di chuyển của các token và có thể khiến một số trạng thái không thể đạt được, ngay cả khi bài toán trên đồ thị vô hướng có thể giải quyết được.

Cho hai tập độc lập I^s và I^t trong một đồ thị có hướng G với $|I^s| = |I^t|$. Một dãy $\langle I_0, I_1, \ldots, I_\ell \rangle$ các tập độc lập của G được gọi là chuỗi tái cấu hình từ I^s đến I^t trong G nếu:

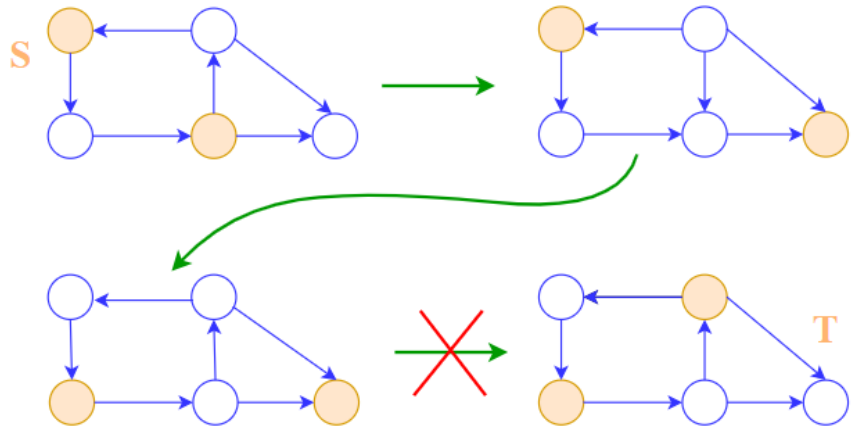
- $I_0 = I^s; \quad I_\ell = I^t$
- Với mọi $i \in [\ell]$, ta có $I_{i-1} \setminus I_i = \{u\}, \quad I_i \setminus I_{i-1} = \{v\}, \quad$ với $(u, v) \in A(G)$.

Độ dài của chuỗi tái cấu hình $\langle I_0, I_1, \ldots, I_\ell \rangle$ là là một dãy các tập độc lập liên tiếp trọng đó mỗi bước $I_i \to I_{i+1}$ tuân theo quy tắc trượt token. Độ dài của chuỗi được định nghĩa là số bước thay đổi tức là ℓ. Khi tồn tại một chuỗi như vậy từ I^s đến I^t ta nói

rằng I^s có thể tái cấu hình thành I^t trong G và ngược lại.

Trong khóa luận này chúng ta sẽ nghiên cứu bài toán trượt token có hướng (Directed Token Sliding). Như vậy bài toán tái cấu hình các tập độc lập trên đồ thị có hướng sẽ được phát biểu như sau:

Liệu có tồn tại một chuỗi các bước hợp lệ để chuyển từ tập độc lập ban đầu I^s sang tập độc lập đích I^t bằng cách trượt token dọc theo các cung có hướng đồng thời đảm bảo rằng các token không bao giờ trở thành láng giềng của nhau tức là các trạng thái trung gian luôn thỏa mãn điều kiện là một tập độc lập ?



Hình 2.2: Token Sliding trên đồ thị có hướng

Ở ví dụ trong **Hình 2.1** (Token Sliding on Undirected Graphs), các token có thể trượt dọc theo các cạnh không có hướng, miễn là tập hợp độc lập được bảo toàn. Tuy nhiên trong bài toán DTS, các token chỉ có thể trượt dọc theo cung theo hướng của nó (từ tail đến head). Điều này làm hạn chế khả năng di chuyển của các token. Cũng vì thế mà tập độc lập trong **Hình 2.2** tái cấu hình thất bại khi mà một token cần di chuyển nhưng không có cung hướng đúng để đi tới trạng thái *T*.

Chương 3

SỰ PHỨC TẠP CỦA BÀI TOÁN DIRECTED TOKEN SLIDING TRÊN CÁC DẠNG ĐỒ THỊ

Bài toán Directed Token Sliding hay trượt token có hướng không phải lúc nào cũng dễ dàng trên mọi loại đồ thị. Trong thực tế, khi xét trên đồ thị có hướng tổng quát, bài toán nhanh chóng trở nên khó giải, thậm chí thuộc lớp PSPACE-complete. Điều này có nghĩa là không tồn tại thuật toán hiệu quả để giải quyết bài toán. Do đó để tiếp cận bài toán, một cách hiệu quả hơn, cần phải tìm một lớp đồ thị có cấu trúc đặc biệt giúp giảm thiệu độ phức tạp của bài toán.

Trong phần này, tôi sẽ trình bày các kết quả về độ phức tạp của bài toán DTS. Cụ thể tôi sẽ chứng minh bài toán này là PSPACE-complete trên đồ thị có hướng (oriented graphs) và NP-complete trên đồ thị có hướng không chu trình (DAGs) từ đó rút ra kết luận tại sao nên ưu tiên làm việc trên polytree với bài toán DTS.

3.1 Bài toán Directed Token Sliding trên oriented graphs

3.1.1 Thuộc tính PSPACE của bài toán DTS trên oriented graphs

Trước tiên, ta cần chứng minh rằng bài toán DTS thuộc lớp PSPACE. Có thể thấy rằng chúng ta có thể kiểm tra xem có tồn tại một chuỗi tái cấu hình hợp lệ hay không bằng cách sử dụng một thuật toán phiếm định (non-deterministic algorithm).Cụ thể hơn, bằng cách đoán phiếm định (“non-deterministically”) một bước đi hợp lệ tại mỗi thời điểm trong quá trình tái cấu hình, ta có thể kiểm tra xem có tồn tại một chuỗi hợp lệ hay không. Theo định lý Savitch [6] bất kỳ bài toán nào có thể giải quyết bằng bộ nhớ phiếm định đa thức thì cũng có thể giải quyết trong bộ nhớ xác định đa thức, nghĩa là bài toán thuộc PSPACE.

Tiếp theo cần chứng minh bài toán DTS là PSPACE-hard bằng cách giảm đa thức từ một bài toán đã biết là thuộc PSPACE-complete. Bài toán TS trên đồ thị vô hướng (undirected Token Sliding), đã được chứng minh là thuộc lớp PSPACE-

complete [40] từ đó có thể xây dựng được một quy giảm đa thức từ TS trên đồ thị vô hướng sang DTS trên đồ thị có hướng. Từ đó ta sẽ chứng minh được định lý:

Định lý 3.1. *Bài toán Directed Token Sliding là PSPACE-complete trên oriented graphs.*

3.1.2 Xây dựng phép quy giảm TS → DTS

Chứng minh. Cho một trường hợp (G, I^s, I^t) TS trên một đồ thị vô hướng G với tập đỉnh $V(G)$ và tập cạnh $E(G)$. Ta cần xây dựng một (G, I^s, I^t) DTS tương ứng trên một oriented graph G'.

Với mỗi đỉnh $v \in V(G)$, ta tạo hai đỉnh v_1, v_2 trong $V(G')$. Thêm một cung có hướng (v_1, v_2) vào $A(G')$, nghĩa là ta buộc phải đi theo hướng từ v_1 đến v_2. Với mỗi cạnh vô hướng $\{u, v\} \in E(G)$, ta thêm bốn cung có hướng vào G':

$$(u_1, v_1), \quad (v_1, u_2), \quad (u_2, v_2), \quad (v_2, u_1).$$

Kết quả là đồ thị G' thu được là một oriented graph, tức là một đồ thị có hướng mà mọi đỉnh chỉ có bậc vào hoặc bậc ra, không có cạnh vô hướng.

Tập token ban đầu được đặt tại $J^s = \{v_1 \mid v \in I^s\}$, tức là lấy mỗi đỉnh v trong tập độc lập ban đầu của G và ánh xạ nó vào v_1 trong đồ thị G'. Tương tự, tập token đích được đặt tại $J^t = \{v_1 \mid v \in I^t\}$. Chúng ta cần chứng minh phát biểu sau:

(G', J^s, J^t) là một yes-instance của DTS khi và chỉ khi (G, I^s, I^t) là một yes-instance của TS.

Chứng minh chiều xuôi

Giả sử (G, I^s, I^t) là một *yes-instance* của TS, nghĩa là tồn tại một chuỗi di chuyển hợp lệ để biến tập độc lập I^s thành I^t. Một dãy tái cấu hình hợp lệ $\langle I_0, I_1, \ldots, I_\ell \rangle$ với $I_0 = I^s$ và $I_\ell = I^t$. Với mỗi bước $I_i \to I_{i+1}$, ta có một token dịch chuyển từ u sang v trên một cạnh vô hướng $\{u, v\}$.

Trong đồ thị G', nếu tồn tại cung (u_1, v_1), ta có thể trượt token từ u_1 đến v_1 theo hướng này, đảm bảo rằng dãy trạng thái trong G' vẫn là tập độc lập. Nếu không tồn tại cung (u_1, v_1), ta có thể di chuyển theo trình tự như sau để tận dụng các cung có hướng đã thêm vào:

$$(u_1 \to v_2 \to u_2 \to v_1),$$

Như vậy, ta có thể xây dựng một dãy tái cấu hình từ J^s đến J^t trong G', chứng tỏ (G', J^s, J^t) là một *yes-instance* của bài toán DTS trên oriented graph.

Chứng minh chiều ngược

Giả sử (G', J^s, J^t) là một *yes-instance* của bài toán DTS, nghĩa là tồn tại một chuỗi di chuyển hợp lệ để biến tập J^s thành J^t.

Nếu một token dịch chuyển từ u_1 đến v_1 trong G', ta có thể ánh xạ bước này thành việc di chuyển token từ u đến v trong G. Vì mọi bước di chuyển trong G' đều tuân theo các cung đã được tạo ra từ cạnh vô hướng $\{u, v\}$ của G, ta có thể chuyển đổi lại thành một chuỗi tái cấu hình hợp lệ trong G.

Do đó, (G, I^s, I^t) cũng là một *yes-instance* của bài toán TS. Từ đó ta đã kết luận được rằng theo định nghĩa của lớp PSPACE-complete, bài toán DTS trên oriented graphs là PSPACE-complete.

3.1.3 Ví dụ minh hoạ phép quy giảm TS → DTS

Xét đồ thị vô hướng $G = (V, E)$ với:

$$V = \{a, b, c, d, e\}, \quad E = \{\{a, b\}, \{b, c\}, \{c, d\}, \{d, a\}\} \cup \{\{d, e\}, \{c, e\}\}.$$

Đây là hình vuông $a - b - c - d - a$ gắn thêm tam giác $d - e - c.$. Giả sử tập token ban đầu và tập token đích là:

$$I^s = \{a, c\}, \quad I^t = \{b, d\}.$$

Ta thấy rằng cả I^s và I^t đều là các tập độc lập trong G. Chuỗi trượt token ngắn nhất khả dĩ để chuyển từ I^s sang I^t là:

Bước	**Cấu hình**	**Lý do vẫn độc lập**
0	$\{a, c\} = I_s$	—
1	$\{a, e\} \quad c \to e$	a không kề e
2	$\{b, e\} \quad a \to b$	b không kề e
3	$\{b, d\} = I_t \quad e \to d$	b không kề d

Bảng 3.1: Chuỗi bước trượt token từ I_s đến I_t

Tiếp theo ta thiết lập một *instance* của bài toán *Directed Token Sliding (DTS)* như sau. Trước hết, với mỗi đỉnh $v \in V$ của đồ thị ban đầu G, ta tạo hai đỉnh v_1 và v_2 trong đồ thị mới G', sau đó thêm cung "tiến" (v_1, v_2) để đảm bảo chiều trượt hợp lệ. Khi đó, tập đỉnh của G' là:

$$V(G') = \{a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2, e_1, e_2\}.$$

Tiếp theo, với mỗi cạnh vô hướng $\{u, v\} \in E$ trong G, ta thay thế bằng bốn cung có hướng trong G':

$$(u_1, v_1), \quad (u_1, v_2), \quad (u_2, v_2), \quad (u_2, v_1).$$

Nếu đồ thị ban đầu có 6 cạnh, thì tổng cộng có $6 \times 4 = 24$ cung mới sinh ra, cộng thêm 5 cung "tiến" (v_1, v_2) từ các đỉnh ban đầu. Cuối cùng, ta thiết lập vị trí token ban đầu và đích như sau:

$$J^s = \{a_1, c_1\}, \quad J^t = \{b_1, d_1\}.$$

Như vậy, đồ thị G' thu được là một *oriented graph* (tức là đồ thị có hướng không chứa cặp cung ngược chiều giữa hai đỉnh).

Cung tiến	**{a,b}**	**{b,c}**	**{c,d}**	**{d,a}**	**{d,e}**	**{c,e}**
$(\mathbf{a_1}, \mathbf{a_2})$	(a_1, b_1)	(b_1, c_1)	(c_1, d_1)	(d_1, a_1)	(d_1, e_1)	(c_1, e_1)
$(\mathbf{b_1}, \mathbf{b_2})$	(b_1, a_2)	(c_1, b_2)	(d_1, c_2)	(a_1, d_2)	(e_1, d_2)	(e_1, c_2)
$(\mathbf{c_1}, \mathbf{c_2})$	(a_2, b_2)	(b_2, c_2)	(c_2, d_2)	(d_2, a_2)	(d_2, e_2)	(c_2, e_2)
$(\mathbf{d_1}, \mathbf{d_2})$	(b_2, a_1)	(c_2, b_1)	(d_2, c_1)	(a_2, d_1)	(e_2, d_1)	(e_2, c_1)
$(\mathbf{e_1}, \mathbf{d_2})$	–	–	–	–	–	–

Bảng 3.2: Danh sách các cung trong đồ thị G sau khi quy đổi từ G

Tiếp theo ta sẽ mô phỏng chuỗi DTS bên trong G'. Hai token ban đầu được đặt tại a_1 và c_1. Chuỗi trượt token dưới đây minh họa quá trình chuyển từ tập độc lập ban đầu $J^s = \{a_1, c_1\}$ sang tập đích $J^t = \{b_1, d_1\}$ trong đồ thị có hướng G', được xây dựng từ đồ thị gốc G.

Ở bước đầu tiên, token tại c_1 được trượt đến e_1 qua cung (c_1, e_1), phát sinh từ cạnh $\{c, e\}$ trong G; vì a không kề e nên a_1 và e_1 không có cung trong G', do đó trạng thái là hợp lệ. Tiếp theo, token tại a_1 trượt đến b_1 qua cung (a_1, b_1), ứng với cạnh $\{a, b\}$; do b và e không kề nhau nên trạng thái vẫn hợp lệ. Cuối cùng, token tại e_1 trượt sang d_1 qua cung (e_1, d_1), tương ứng với cạnh $\{d, e\}$; vì b không kề d nên đây vẫn là một tập độc lập. Toàn bộ ba bước trên duy trì tính chất không kề nhau tại mọi thời điểm, do đó, (G', J^s, J^t) là một *yes-instance* hợp lệ của bài toán Directed Token Sliding.

Khi muốn di chuyển token ngược chiều cạnh gốc, chẳng hạn từ $b \to a$, trong đồ thị G' không tồn tại cung trực tiếp (b_1, a_1). Do đó, ta sử dụng một cấu trúc phụ (gadget) gồm ba cung để mô phỏng bước đi này:

$$b_1 \to a_2 \to b_2 \to a_1$$

Tại thời điểm thực hiện, hai đỉnh trung gian a_2 và b_2 chắc chắn rỗng, vì mỗi cặp đỉnh (v_1, v_2) chỉ chứa tối đa một token. Điều này đảm bảo tính độc lập luôn được duy trì trong suốt quá trình trượt.

Tổng quát, để mô phỏng một bước ngược chiều với cạnh $\{v, u\}$, ta sử dụng chuỗi ba cung:

$$v_1 \to u_2 \to v_2 \to u_1$$

Ngược lại, nếu có cung thuận (v_1, u_1), ta chỉ cần thực hiện một bước trượt đơn giản. Các cung "tiến" dạng (v_1, v_2) không được sử dụng để mô phỏng chuỗi trượt, mà chỉ nhằm đảm bảo G' là một *oriented graph* – tức là không có cặp cung ngược chiều giữa hai đỉnh.

Như vậy, ta đã tái hiện đầy đủ cả chiều thuận và chiều ngược của chuỗi trượt token từ đồ thị gốc sang đồ thị G' mà vẫn đảm bảo các điều kiện của bài toán *Directed Token Sliding*.

3.2 Bài toán Directed Token Sliding trên DAGs

3.2.1 Thuộc tính NP của bài toán DTS trên DAGs

DAG - Directed Acyclic Graph đơn giản là một đồ thị có hướng nhưng không có chu trình. Đầu tiên ta sẽ chứng minh bài toán DTS trên DAG thuộc lớp NP. Sau đó, ta chứng minh rằng bài toán này NP-hard bằng cách xây dựng một phép giảm từ bài toán Multicolored Independent Set, một bài toán đã biết là NP-complete. Từ đó sẽ chứng minh được đinh lý sau:

Định lý 3.2. *Bài toán Directed Token Sliding trên DAGs là NP-complete khi tham số hóa theo số lượng token.*

Chứng minh. Một bài toán thuộc NP nếu ta có thể kiểm tra một lời giải hợp lệ trong thời gian đa thức. Trong trường hợp của DTS trên DAGs, giả sử ta có một dãy tái cấu hình hợp lệ:

$$I_0 \to I_1 \to I_2 \to \cdots \to I_\ell$$

với $I_0 = I^s$ và $I_\ell = I^t$. Trong đồ thị G là một đồ thị có hướng không chu trình (DAG), mỗi token chỉ có thể di chuyển theo hướng của cung mà không thể quay lại đỉnh đã đi qua. Vì vậy, mỗi token thực hiện tối đa $O(|V|)$ bước. Nếu có tối đa $|V|$ tokens, tổng số bước di chuyển bị chặn trên bởi $O(|V|^2)$. Điều này đảm bảo rằng mọi chuỗi tái cấu

hình hợp lệ đều có độ dài tối đa $O(|V|^2)$, chứng minh rằng bài toán thuộc NP vì một lời giải có thể được kiểm tra trong thời gian đa thức.

Tiếp theo ta chứng minh NP-hard bằng cách quy về từ Multicolored Independent Set. Multicolored Independent Set (MIS) là bài toán đã biết là NP-complete [41], trong đó, với một đồ thị vô hướng $G = (V, E)$, một số nguyên k, và một phân hoạch $\{V_1, V_2, \ldots, V_k\}$ của tập đỉnh V, ta cần xác định xem có tồn tại một tập độc lập $X \subseteq V$ sao cho $|X| = k$ và X chứa đúng một đỉnh từ mỗi V_i.

3.2.2 Xây dựng phép quy giảm MIS → DTS

Cho một *instance* $(G, k, \{V_1, \ldots, V_k\})$ của bài toán Multicolored Independent Set, ta xây dựng một *instance* (G', I^s, I^t) của bài toán DTS sao cho G' là một DAG và $|I^s| = |I^t| = k + 1$. Từ đó ta xây dựng một *instance* (G', I^s, I^t) của bài toán DTS trên DAG như sau:

Tập đỉnh $V(G')$:

$$V(G') = U \cup V(G) \cup W$$

Trong đó:

- $U = \{u_1, u_2, \ldots, u_k, u_{k+1}\}$ (tập đỉnh ban đầu chứa token).
- $W = \{w_1, w_2, \ldots, w_k, w_{k+1}\}$ (tập đỉnh đích).
- $V(G)$ là tập đỉnh của đồ thị ban đầu.

Tập cung $A(G')$:

$$A(G') = A_{UW} \cup A_{UV} \cup A_{VW} \cup A_V$$

Trong đó:

- A_{UW}: Tạo một cung (u, w) cho mọi $u \in U, w \in W$.

$$A_{UW} = \{(u_i, w_j) \mid 1 \leq i, j \leq k + 1\}$$

- A_{UV}: Nối mỗi u_i với tất cả các đỉnh trong V_i.

$$A_{UV} = \{(u_i, v) \mid v \in V_i, 1 \leq i \leq k\}$$

- A_{VW}: Nối mỗi $v \in V_i$ với w_i.

$$A_{VW} = \{(v, w_i) \mid v \in V_i, 1 \leq i \leq k\}$$

- A_V: Định hướng các cạnh trong G sao cho G' vẫn là một DAG.

$$A_V = \{(v, v') \mid v \in V_i, v' \in V_j, i < j\}$$

Đây là cách đảm bảo rằng G' vẫn là một DAG, vì các cạnh trong cùng một tập V_i không có hướng ngược lại. Tập token ban đầu $I^s = U = \{u_1, u_2, \ldots, u_k, u_{k+1}\}$, tập token đích $I^t = W = \{w_1, w_2, \ldots, w_k, w_{k+1}\}$. Khi đó ta phát biểu rằng:

G có một tập độc lập đa sắc khi và chỉ khi G' có một chuỗi tái cấu hình hợp lệ.

Chứng minh chiều xuôi

Giả sử $X = \{v_1, v_2, \ldots, v_k\}$ là một tập độc lập đa sắc trong G, tức là mỗi v_i thuộc V_i. Khi đó, ta có thể di chuyển token từ u_i đến v_i theo cung (u_i, v_i) trong A_{UV}. Vì X là tập độc lập, các token không chạm nhau trong quá trình di chuyển.

Tiếp theo, di chuyển token từ u_{k+1} sang w_{k+1}. Cuối cùng, di chuyển các token từ v_i sang w_i theo các cung trong A_{VW}. Vì không có cung nào nối v_i với w_j khi $i \neq j$, tập token vẫn là tập độc lập ở mỗi bước. Do đó, nếu G có một tập độc lập đa sắc kích thước k, thì G' có một chuỗi tái cấu hình hợp lệ từ I^s đến I^t.

Chứng minh chiều ngược

Giả sử tồn tại một chuỗi tái cấu hình hợp lệ từ I^s đến I^t. Lưu ý rằng, để di chuyển token từ $u_1, u_2, \ldots, u_k$ đến W, chúng phải đi qua các đỉnh trong $V(G)$, vì:

- $u_1, u_2, \ldots, u_k$ chỉ có thể đi đến các đỉnh trong $V(G)$ theo cung trong A_{UV}.
- Sau khi vào $V(G)$, các token phải đi đến W thông qua A_{VW}.

Điều này có nghĩa là các token đã chọn đúng một đỉnh từ mỗi tập V_i. Hơn nữa, vì mỗi bước trong chuỗi tái cấu hình phải duy trì tính chất tập độc lập, nên các đỉnh được chọn từ $V_1, V_2, \ldots, V_k$ phải tạo thành một tập độc lập đa sắc trong G. Như vậy, nếu G' có một chuỗi tái cấu hình hợp lệ, thì G có một tập độc lập đa sắc.

Vậy, nếu bài toán Multicolored Independent Set có lời giải, thì bài toán DTS trên DAGs cũng có lời giải và ngược lại. Do Multicolored Independent Set là NP-complete, suy ra DTS trên DAGs cũng NP-hard. Hơn nữa, ta đã chứng minh rằng bài toán nằm trong NP, do đó bài toán này là NP-complete.

3.2.3 Ví dụ minh hoạ phép quy giảm MIS → DTS

Để minh hoạ cách quy giảm từ bài toán Multicolored Independent Set (MIS) sang bài toán Directed Token Sliding (DTS) trên đồ thị có hướng không chu trình (DAG), ta chọn một *instance* đơn giản với $k = 2$ (tức 2 màu).

- $V_1 = \{a_1, a_2\}$ – các đỉnh mang "màu 1",
- $V_2 = \{b_1, b_2\}$ – các đỉnh mang "màu 2".

Các cạnh của đồ thị được định nghĩa như sau:

$$E = \{\{a_1, b_1\}\ \{a_2, b_2\}\}.$$

Mục tiêu của bài toán là tìm một tập độc lập đa sắc kích thước $k = 2$, tức là một tập $X \subseteq V$ sao cho $|X| = 2$, trong đó X chứa đúng một đỉnh từ mỗi tập V_i (tương ứng với mỗi màu), và không có hai đỉnh nào trong X kề cạnh nhau trong đồ thị G.

Nếu chọn tất cả các đỉnh có chỉ số 1, tức là a_1, b_1, thì hai đỉnh này tạo thành một đường thẳng bị chặn 2 đầu, do đó không thỏa mãn điều kiện của một tập độc lập. Tương tự, nếu chọn các đỉnh có chỉ số 2 như a_2, b_2, thì chúng cũng tạo thành một đường thẳng và không hợp lệ. Tuy nhiên, nếu ta chọn $X = \{a_1, b_2\}$ thì đây là một tập thỏa mãn tất cả các yêu cầu: mỗi đỉnh thuộc một màu khác nhau, và không có cạnh nào nối giữa bất kỳ hai đỉnh nào trong tập. Do đó, X là một tập độc lập đa sắc hợp lệ. Vì vậy, đây là một *yes-instance* của bài toán Multicolored Independent Set.

Tiếp theo chúng ta cần xây dựng đồ thị có hướng không chu trình $G' = (V', A')$. Để chuyển hoá một *instance* của Multicolored Independent Set thành một *instance* của Directed Token Sliding trên DAG, trước hết ta cần tổ chức tập đỉnh của đồ thị mới G' thành ba miền riêng biệt, mỗi miền đảm nhận một vai trò rõ ràng trong quá trình tái cấu hình token.

1. **Miền khởi phát** U:

   $$U = \{u_1, u_2, u_3\}$$

   Đây là tập gồm $k + 1 = 3$ đỉnh, tương ứng với bốn token xuất phát. Hai token đầu (u_1, u_2) sẽ phải "đi qua" các lớp màu V_1, V_2 để lựa chọn đỉnh thích hợp. Token thứ ba (u_3) đóng vai trò "đối trọng": nó di chuyển thẳng tới miền đích để giữ nguyên kích thước $k + 1$ của tập độc lập trong suốt quá trình.
2. **Miền trung gian** V:

   $$V = V_1 \cup V_2 = \{a_1, a_2, b_1, b_2\}$$

Mỗi V_i đại diện cho một "màu" trong bài toán MIS gốc—tức là một lớp của phân hoạch đỉnh:

$$V_1 = \{a_1, a_2\}, \quad V_2 = \{b_1, b_2\}$$

Yêu cầu "đa sắc" sẽ được thực thi bằng cách buộc mỗi token u_i (với $1 \leq i \leq 3$) phải ghé đúng một đỉnh trong lớp V_i trước khi tiếp tục hành trình.

3. **Miền đích** W:

$$W = \{w_1, w_2, w_3\}$$

Giống như miền khởi phát, miền này cũng gồm $k + 1$ đỉnh và là điểm dừng cuối cùng của mọi token. Cấu trúc cung sẽ ép mỗi token xuất phát từ V_i chỉ được phép kết thúc tại w_i cùng chỉ số, qua đó bảo toàn tính độc lập.

Tổng hợp lại, tập đỉnh đầy đủ của đồ thị chuyển đổi là

$$V' = U \cup V \cup W$$

Việc tách V' thành ba miền $U - V - W$ không chỉ giúp trực quan hoá quá trình di chuyển (khởi hành $\rightarrow$ chọn màu $\rightarrow$ về đích) mà còn là chìa khoá chứng minh: mọi chuỗi tái cấu hình trong G' tương ứng một-một với một tập độc lập đa sắc cỡ k trong đồ thị gốc G.

Sau khi đã cố định ba miền đỉnh U, V và W của đồ thị $G' = (V', A')$, bước tiếp theo là liệt kê đầy đủ – có hệ thống – toàn bộ các cung (cạnh có hướng) sao cho G' vừa phản ánh chính xác ràng buộc "đa sắc", vừa bảo đảm không tạo chu trình (do đó G' là một DAG). Tập cung sẽ được chia thành bốn nhóm, ký hiệu là (1) đến (4).

1. **Các cung từ miền khởi phát** U **sang miền đích** W
   Để duy trì kích thước tập token luôn bằng $k + 1$ (ở đây là 4), ta cho phép mỗi cặp (u_i, w_j) có một cung trực tiếp. Tập cung này được ký hiệu là:

$$A_{UW} = \{(u_i, w_j) \mid i, j \in \{1, 2, 3\}\}.$$

   Như vậy, nhóm (1) chứa $3 \times 3 = 9$ cung, tạo nên các "đường cao tốc" cho bất kỳ token nào ở U có thể lập tức rẽ vào W khi cần thiết (điển hình là token "phụ" u_3).

2. **Các cung từ** U **sang từng lớp màu trong** V
   Để buộc ba token "chính" lần lượt chọn đúng một đỉnh theo màu, ta kết nối:

$$u_1 \rightarrow \{a_1, a_2\}, \quad u_2 \rightarrow \{b_1, b_2\}.$$

   Nhóm cung này có tổng cộng 4 cung: (u_1, a_1), (u_1, a_2), (u_2, b_1), (u_2, b_2). Mỗi token u_i chỉ được phép ghé qua đúng lớp V_i, qua đó bảo toàn ràng buộc "đa sắc".

3. **Các cung từ V về miền đích W**
   Để khép lại hành trình, mọi đỉnh trong lớp V_i đều được nối đến duy nhất một đỉnh w_i có cùng chỉ số:
   $$a_\bullet \to w_1, \quad b_\bullet \to w_2.$$
   Cụ thể, có 4 cung trong nhóm này: (a_1, w_1), (a_2, w_1), (b_1, w_2), (b_2, w_2). Nhờ vậy, sau khi "chọn màu", mỗi token chỉ có một lối ra duy nhất và không thể di chuyển sang vùng đích của màu khác, từ đó giữ nguyên được tính chất độc lập của tập.
4. **Các cung nội bộ giữa các lớp màu V_1, V_2, V_3**
   Phần còn lại của đồ thị gốc G quyết định các cạnh vô hướng $\{v, v'\}$ giữa các đỉnh khác màu. Để duy trì tính chất có hướng và đảm bảo đồ thị G' là một DAG, ta định hướng các cạnh này theo thứ tự màu tăng dần $1 < 2$, cụ thể:
   $$a_1 \to b_1, \quad a_2 \to b_2.$$
   Như vậy, nhóm (4) đóng góp thêm 2 cung. Việc luôn định hướng "từ màu nhỏ sang màu lớn" giúp loại bỏ hoàn toàn chu trình nội bộ trong miền V, đảm bảo đồ thị G' không chứa chu trình.

Như vậy ta có thể tính toán được tổng số cung đó là:

$$|A'| = \underbrace{9}_{A_{UW}} + \underbrace{4}_{A_{UV}} + \underbrace{4}_{A_{VW}} + \underbrace{2}_{A_V} = 19 \text{ cung}.$$

Vì mọi cung đều tiến xuôi theo trật tự $U \to V \to W$ hoặc (trong V) theo hướng "màu nhỏ $\to$ màu lớn", nên không tồn tại chu trình định hướng nào. Do đó, G' là một đồ thị có hướng không chu trình (DAG).

Tiếp theo ta cần phải khởi tạo vị trí token với tập token ban đầu:

$$I^s = U = \{u_1, u_2, u_3\}.$$

và tập token đích:

$$I^t = W = \{w_1, w_2, w_3\}.$$

Cấu trúc ba miền cùng với 19 cung đã được mô tả ở trên bảo đảm rằng bất kỳ chuỗi tái cấu hình hợp lệ nào đưa I^s đến I^t đều phải "ép" ba token chính lựa chọn đúng một đỉnh trong mỗi lớp V_i. Điều này tái hiện chính xác điều kiện của một tập độc lập đa sắc độ lớn k trong bài toán gốc.

Tiếp theo ta sẽ mô phỏng lời giải MIS với mục tiêu là xuất phát với tập token $I^s = U$ và bằng cách dựa trên lời giải đa sắc $X = \{a_1, b_2\}$, chứng minh rằng ta có thể

di chuyển từng token tới miền đích W mà luôn giữ được tính chất độc lập trong suốt quá trình tái cấu hình.

Bước	**Cung (hướng đi)**	**Vị trí token sau bước**	**Vì sao vẫn là tập độc lập**
0	— (trạng thái đầu)	$\{u_1, u_2, u_3\}$	Ba đỉnh $u_1, u_2, u_3 \in U$; không có cạnh nội bộ trong U.
1	$u_1 \to a_1$	$\{a_1, u_2, u_3\}$	Chỉ một token vừa vào V; trong V hiện có 1 token nên không thể xung đột.
2	$u_2 \to b_2$	$\{a_1, b_2, u_3\}$	$a_1 \in V_1$, $b_2 \in V_2 \Rightarrow$ khác màu, giữa a_1 và b_2 không có cạnh.
3	$b_2 \to w_2$	$\{a_1, w_2, u_3\}$	$w_2 \in W$; các đỉnh trong W không kề nhau và tách biệt khỏi V.
4	$u_3 \to w_3$	$\{a_1, w_2, w_3\}$	Chỉ còn 1 token trong V (a_1); các token ở W luôn độc lập với a_1.
5	$a_1 \to w_1$	$\{w_1, w_2, w_3\} = I_t$	W không chứa cạnh nội bộ $\Rightarrow$ tập cuối vẫn độc lập.

Bảng 3.3: Chuỗi tái cấu hình DTS mô phỏng lời giải MIS $X = \{a_1, b_2\}$ ($k = 2$, 3 token)

Như vậy, thông qua ví dụ cụ thể của phép quy giảm MIS về DTS, ta đã mô phỏng đầy đủ cách ánh xạ giữa lời giải của bài toán *Multicolored Independent Set* và *Directed Token Sliding*. Việc này không chỉ giúp chứng minh tính tương đương logic giữa hai bài toán, mà còn minh hoạ rõ ràng cách mà chuỗi tái cấu hình trong G' phản ánh chính xác một tập độc lập đa sắc trong đồ thị gốc G.

Chương 4

BÀI TOÁN DIRECTED TOKEN SLIDING TRÊN POLYTREE

Ở phần trước chúng ta đã chứng minh được bài toán DTS là PSPACE-complete trên oriented graphs. Điều này có nghĩa là bài toán thuộc vào nhóm các vấn đề cực kỳ khó, với không chỉ độ phức tạp cao mà còn khó giải quyết ngay cả khi có thuật toán chạy trong thời gian đa thức không xác định trước (non-deterministic polynomial space). Nguyên nhân chính là do có quá nhiều cách di chuyển token, cũng như có thể tồn tại các đường vòng phức tạp gây khó khăn trong việc kiểm tra tính khả thi của việc tái cấu hình. Vậy nên DTS trên oriented graph không phù hợp để giải quyết hiệu quả.

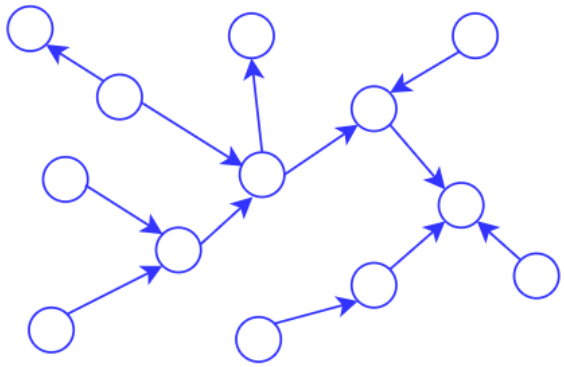
So với oriented graphs, DAGs đã đơn giản hơn một bậc về mặt cấu trúc vì không có chu trình, nhưng vẫn có thể có nhiều nhánh phân rã phức tạp. Bài toán DTS trên DAGs được chứng minh là NP-complete, tức là vẫn có độ khó rất cao. Một trong những lý do khiến bài toán vẫn khó trên DAGs là vì một đồ thị DAG có thể có nhiều đường đi giữa hai đỉnh, dẫn đến rất nhiều cách di chuyển token mà ta phải kiểm tra. Ngoài ra, DAGs vẫn có thể có các nhánh lớn và phân rã, làm cho việc xác định một chuỗi tái cấu hình hợp lệ trở nên không rõ ràng và phải thử nhiều trường hợp. Mặc dù bài toán đã nằm trong phạm vi NP, tức là có thể kiểm tra tính hợp lệ của một lời giải trong thời gian đa thức nhưng điều này vẫn chưa đủ để giải quyết bài toán một cách hiệu quả.

Một đồ thị có hướng G được gọi là một polytree nếu đồ thị vô hướng nền của nó là một cây (tree), tức là một đồ thị vô hướng liên thông không chứa chu trình. Điều này có nghĩa là nếu ta bỏ qua hướng của các cung trong G, ta sẽ thu được một cây, đảm bảo rằng giữa mọi cặp đỉnh trong đồ thị tồn tại duy nhất một đường đi không có chu trình.

Một polytree cũng có thể được xem là một trường hợp đặc biệt của một DAGs. Tuy nhiên, không phải mọi DAG đều là polytree, vì một DAG tổng quát có thể có nhiều đường đi khác nhau giữa hai đỉnh bất kỳ, trong khi đối với polytree, giữa bất kỳ hai đỉnh nào luôn tồn tại chính xác một đường đi duy nhất theo hướng của cung, đảm bảo

rằng đồ thị không chứa bất kỳ chu trình nào.

Điều này giúp loại bỏ sự phức tạp của các đường đi thay thế trong DAG, dẫn đến một cấu trúc rõ ràng và dễ kiểm soát hơn. Nhờ tính chất này, ta có thể xác định ngay đường đi duy nhất mà một token phải di chuyển theo, từ đó giảm thiểu số trường hợp cần kiểm tra.



Hình 4.1: Polytree

Định lý 4.1. *Cho $T = (V, A)$ là một polytree. Bài toán Directed Token Sliding trên T có thể được giải quyết trong thời gian tuyến tính $O(|V|)$. Hơn nữa, nếu tồn tại một chuỗi tái cấu hình hợp lệ, thì tất cả các chuỗi tái cấu hình hợp lệ đều có cùng độ dài, và ta có thể xây dựng một trong các chuỗi đó trong thời gian $O(|V|^2)$.*

4.1 Ghép các đường đi có hướng trên Polytree

4.1.1 Tồn tại DPM là điều kiện cần để giải bài toán DTS trên Polytree

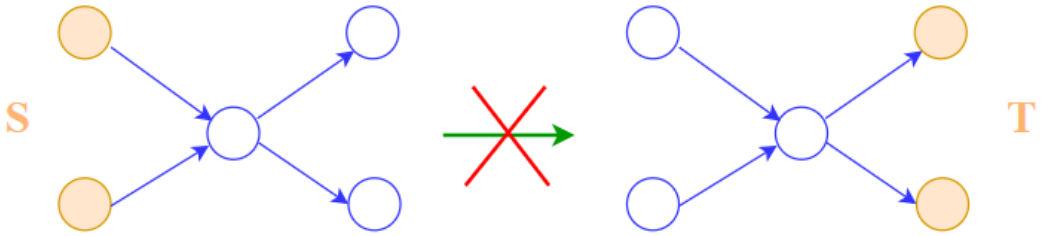
Xét hai tập hợp độc lập I^s và I^t trong một *polytree* $T = (V, A)$, thỏa mãn điều kiện $|I^s| = |I^t|$. Nếu I^s có thể được tái cấu hình thành I^t, thì trong một chuỗi tái cấu hình hợp lệ, mỗi token i sẽ di chuyển dọc theo một đường đi có hướng P_i. Tập hợp các đường đi này được ký hiệu là:

$$\mathcal{P} := \{P_1, P_2, \ldots, P_k\}.$$

Tập hợp P này tạo thành *phép ghép các đường đi có hướng* (Directed Path Matching-DPM) từ I^s đến I^t. Sự tồn tại của một phép ghép các đường đi có hướng là một điều kiện cần hiển nhiên để bài toán có lời giải. Nếu không tồn tại một tập hợp các đường đi như vậy, rõ ràng ta không thể di chuyển tất cả các token từ I^s đến I^t một

cách hợp lệ.

Mặc dù sự tồn tại của *DPM là điều kiện cần nhưng nó không phải điều kiện đủ*. Để minh họa , hãy xem xét một đồ thị có hướng với đồ thị vô hướng nền là một ngôi sao có bốn lá trong đó đỉnh trung tâm có hai đỉnh vào và hai đỉnh ra.



Hình 4.2: Đồ thị hình ngôi sao

Chọn tập I^s là hai đỉnh vào và I^t là hai đỉnh ra của đỉnh trung tâm. Trong trường hợp này, có thể tìm thấy một *DPM* giữa I^s và I^t tức là tồn tại đường đi có hướng cho từng đỉnh trong tập độc lập. Tuy nhiên, như trong **Hình 4.2** sẽ không có lời giải vì không có cách hợp lệ để di chuyển các token mà không vi phạm quy tắc độc lập của nghiệm trung gian. Ví dụ này cho thấy rằng việc tồn tại một phép ghép các đường đi có hướng không đảm bảo rằng các token có thể thực sự di chuyển theo một chuỗi tái cấu hình hợp lệ.

Mặc dù không phải điều kiện đủ, nhưng DPM vẫn cho ta vẫn cho ta những lợi ích to lớn trong bài toán DTS trên polytree. Do mỗi cặp đỉnh trong polytree chỉ có một đường đi duy nhất việc kiểm tra DPM sẽ giúp xác định xem có thể di chuyển tất cả các token từ I^s đến I^t hay không. Cùng với đó việc đặc trưng hóa DPM sẽ giúp xây dựng một thuật toán tuyến tính để kiểm tra tính tái cấu hình.

4.1.2 Xây dựng hàm trọng số để xác định DPM trên Polytree

Cho hai tập đỉnh X và Y (không nhất thiết phải rời nhau) của một polytree $T = (V, A)$, với điều kiện $|X| = |Y|$. Ta muốn tìm một ánh xạ song ánh $\pi : X \to Y$ sao cho mỗi đỉnh $x \in X$ được nối đến $\pi(x) \in Y$ bằng một đường đi có hướng trong T.

Vì T là một polytree, nên đối với mỗi đỉnh $x \in X$, tồn tại duy nhất một đường đi vô hướng từ x đến $\pi(x)$, ký hiệu là P_x. Nếu ta định hướng tất cả các cung trên P_x theo chiều từ x đến $\pi(x)$, ta thu được một đường đi có hướng $P_x^{\rightarrow}$. Từ đó, ta định nghĩa một hàm trọng số mô tả sự chênh lệch số lần một cung e được đi theo hai hướng trong

các đường đi $P_x^{\rightarrow}$.

Định nghĩa hàm trọng số $w(e; X, Y, \pi)$

Với mỗi cung $e \in A$, ta định nghĩa hàm trọng số:

$$w(e; X, Y, \pi) = |\{x \in X \mid e \in P_x^{\rightarrow}\}| - \left|\{x \in X \mid e^{-1} \in P_x^{\rightarrow}\}\right|$$

- $P_x^{\rightarrow}$ là đường đi có hướng từ x đến $\pi(x)$.
- e^{-1} là cung ngược của e.

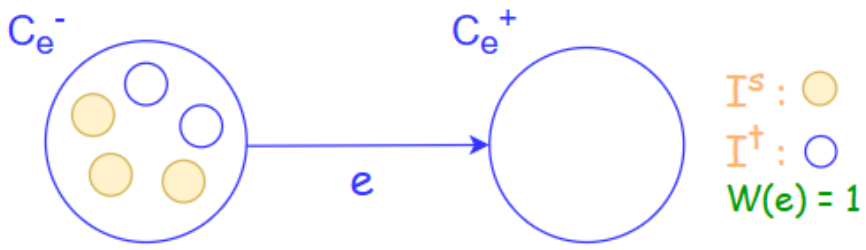
Hàm $w(e; X, Y, \pi)$ đo lường sự chênh lệch giữa số lượng đường đi có hướng $P_x^{\rightarrow}$ đi qua e theo chiều thuận và số lượng đường đi có hướng $P_x^{\rightarrow}$ đi qua e chiều ngược lại.

Xét một cung $e = (u, v) \in A$. Nếu loại bỏ cung e ta sẽ thu được cây con T' được tạo ra bằng cách loại bỏ cung e khỏi polytree T. Khi đó, T' có hai thành phần liên thông yếu được xác định như sau:

- C_e^- là tập đỉnh trong thành phần liên thông yếu của T' chứa đỉnh đầu của cung e (tập các đỉnh có thể đi tới u mà không cần đi qua cung e).
- C_e^+ là tập đỉnh trong thành phần liên thông yếu của T' chứa đỉnh cuối của cung e (tập các đỉnh có thể đi tới v mà không cần đi qua cung e).

Xét một đỉnh $x \in X$, đường đi có hướng $P_x^{\rightarrow}$ từ x đến $\pi(x)$ có thể sử dụng cung e theo một trong hai cách sau:

- **Sử dụng** e **theo chiều thuận**: Điều này xảy ra khi $x \in C_e^-$ và $\pi(x) \in C_e^+$, tức là đường đi $P_x^{\rightarrow}$ từ x đến $\pi(x)$ đi qua cung e theo đúng hướng của cung.
- **Sử dụng** e **theo chiều ngược lại**: Điều này xảy ra khi $x \in C_e^+$ và $\pi(x) \in C_e^-$, tức là đường đi $P_x^{\rightarrow}$ đi qua cung e nhưng theo hướng ngược lại.



Hình 4.3: Minh họa cách xác định trọng số

Từ hai trường hợp trên, ta có thể biểu diễn hàm trọng số như sau:

$$w(e; X, Y, \pi) = \left|\{x \in X \mid x \in C_e^-, \pi(x) \in C_e^+\}\right| - \left|\{x \in X \mid x \in C_e^+, \pi(x) \in C_e^-\}\right|$$

Do ánh xạ π là song ánh từ X đến Y, số phần tử của X thuộc C_e^- nhưng ánh xạ đến C_e^+ chính là số đỉnh của X nằm trong C_e^-, tương tự với Y. Từ đó, ta có:

$$w(e; X, Y, \pi) = |C_e^- \cap X| - |C_e^- \cap Y|$$

Kết quả này cho ta thấy một tính chất quan trọng là giá trị của $w(e; X, Y, \pi)$ không phụ thuộc vào ánh xạ π mà chỉ phụ thuộc vào các tập X và Y, tức là với mọi ánh xạ song ánh $\pi : X \to Y$, giá trị của $w(e; X, Y, \pi)$ là như nhau.

Ta có thể định nghĩa hàm trọng số một cách tổng quát như sau:

$$w(e; X, Y) := |C_e^- \cap X| - |C_e^- \cap Y|, \quad \forall e \in A.$$

Từ đây, ta có thể sử dụng hàm $w(e; X, Y)$ để xác định các điều kiện cần và đủ cho sự tồn tại của một phép ghép các đường đi có hướng trong polytree.

4.1.3 Điều kiện cần và đủ để tồn tại DPM trên Polytree

Ở phần trước, chúng ta đã định nghĩa được hàm trọng số $w(e; X, Y, \pi)$ nhằm đo lường sự chênh lệch giữa số lượng đường đi có hướng từ tập X đến tập Y trên một polytree. Một tính chất quan trọng đã được chứng minh là giá trị của $w(e; X, Y, \pi)$ không phụ thuộc vào ánh xạ song ánh π cụ thể, mà chỉ phụ thuộc vào hai tập đỉnh X và Y. Điều này cho phép ta định nghĩa một cách tổng quát hàm trọng số $w(e; X, Y)$ mà không cần xét đến ánh xạ cụ thể.

Tiếp theo, chúng ta sẽ sử dụng hàm trọng số này để xác định điều kiện cần và đủ cho sự tồn tại của một phép ghép các đường đi có hướng giữa hai tập đỉnh trong polytree. Cụ thể, chúng ta sẽ chứng minh rằng điều kiện $w(e; X, Y) \geq 0$ với mọi cung e là điều kiện tương đương với sự tồn tại của một phép ghép các đường đi có hướng. Điều này giúp đơn giản hóa bài toán và đưa ra một tiêu chí rõ ràng để kiểm tra tính khả thi của quá trình tái cấu hình token trên polytree.

Bổ đề 4.2. *Cho hai tập đỉnh X và Y (không nhất thiết phải rời nhau) của một polytree $T = (V, A)$ với điều kiện $|X| = |Y|$. Khi đó, tồn tại một phép ghép các đường đi có hướng (DPM) từ X đến Y khi và chỉ khi với mọi cung $e \in A$, ta có:*

$$w(e; X, Y) \geq 0.$$

Chứng minh điều kiện cần

Giả sử tồn tại một DPM từ tập đỉnh X đến tập đỉnh Y. Điều này có nghĩa là tồn tại một tập các đường đi có hướng:

$$\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$$

trong đó mỗi đường đi P_i bắt đầu từ một đỉnh thuộc X và kết thúc tại một đỉnh thuộc Y. Các đường đi này thỏa mãn các tính chất:

- Không có hai đường đi nào sử dụng chung một đỉnh trung gian.
- Mỗi đường đi chỉ sử dụng các cung theo hướng cho trước trong polytree T.

Giờ ta xét một cung $e = (u, v) \in A$, tức là một cung có hướng từ đỉnh u đến v. Nếu ta loại bỏ cung này khỏi T, thì đồ thị còn lại sẽ bị chia thành hai thành phần liên thông yếu:

- C_e^-: Tập các đỉnh thuộc thành phần liên thông yếu chứa đỉnh đầu u của cung e. Tức là, nếu ta chỉ xét đồ thị sau khi loại bỏ e, thì mọi đường đi từ bất kỳ đỉnh nào trong C_e^- đến u vẫn tồn tại mà không cần sử dụng cung e.
- C_e^+: Tập các đỉnh thuộc thành phần liên thông yếu chứa đỉnh cuối v của cung e. Nghĩa là, nếu ta loại bỏ e, thì các đỉnh trong C_e^+ vẫn có thể đi đến v mà không cần e.

Vì $\mathcal{P}$ là một DPM hợp lệ, mỗi đường đi P_i từ $x \in X$ đến $\pi(x) \in Y$ có thể sử dụng cung e theo hai cách:

- **Sử dụng e theo chiều thuận**: Điều này xảy ra khi đỉnh đầu của đường đi P_i thuộc về tập C_e^- và đỉnh cuối của nó thuộc tập C_e^+.
  Tức là, nếu $x \in C_e^-$ và $\pi(x) \in C_e^+$, thì đường đi P_i sẽ sử dụng cung e đúng theo hướng của nó trong đồ thị.
- **Sử dụng e theo chiều ngược lại**: Điều này xảy ra khi đỉnh đầu của đường đi P_i thuộc về tập C_e^+ và đỉnh cuối của nó thuộc tập C_e^-.
  Tức là, nếu $x \in C_e^+$ và $\pi(x) \in C_e^-$, thì đường đi P_i sẽ đi qua cung e nhưng theo hướng ngược lại.

Do tất cả các đường đi trong $\mathcal{P}$ là hợp lệ trong đồ thị có hướng T, nghĩa là chúng chỉ có thể đi theo hướng của các cung đã cho. Điều này có nghĩa là số lượng đường đi đi qua e theo chiều thuận phải nhiều hơn hoặc bằng số lượng đường đi đi qua e theo

chiều ngược lại. Do đó:

$$\left|C_e^- \cap X\right| \geq \left|C_e^- \cap Y\right| \quad \Rightarrow \quad w(e; X, Y) = \left|C_e^- \cap X\right| - \left|C_e^- \cap Y\right| \geq 0.$$

Điều này chứng tỏ rằng nếu tồn tại một phép ghép các đường đi có hướng hợp lệ từ X đến Y, thì với mọi cung $e \in A$, trọng số $w(e; X, Y)$ phải thỏa mãn điều kiện:

$$w(e; X, Y) \geq 0.$$

Ta vừa chứng minh rằng nếu tồn tại một phép ghép các đường đi có hướng, thì trọng số $w(e; X, Y)$ của mọi cung e trong A đều không âm. Đây chính là điều kiện cần của bài toán.

Chứng minh điều kiện đủ

Ngược lại, giả sử với mọi cung $e \in A$, ta có:

$$w(e; X, Y) \geq 0.$$

Ta sẽ chứng minh rằng tồn tại một DPM từ X đến Y bằng quy nạp trên tổng giá trị:

$$S = \sum_{e \in A} w(e; X, Y).$$

Bước cơ sở: $S = 0$, lại có $w(e; X, Y) \geq 0 \; \forall e \in A$, suy ra $w(e; X, Y) = 0$. Khi đó:

$$|C_e^- \cap X| = |C_e^- \cap Y|, \quad \forall e \in A.$$

Điều này suy ra $X = Y$, do đó ta có thể xây dựng phép ghép các đường đi bằng cách ánh xạ mỗi $x \in X$ tới chính nó trong Y, đảm bảo sự tồn tại của DPM.

Bước quy nạp ($S > 0$): Giả sử kết luận đúng với mọi tập X, Y sao cho $S \leq k$, ta chứng minh điều đó đúng cho $S = k + 1$.

Vì $w(e; X, Y) \geq 0$ với mọi $e \in A$, tồn tại cung $e^* \in A$ sao cho: $w(e^*; X, Y) > 0$. T' là thành phần liên thông yếu của đồ thị con được tạo ra bằng cách chỉ lấy các cung có $w(e'; X, Y) > 0$, trong đó $e^* \in A$ là một cung thuộc T'. Khi đó, số lượng đỉnh thuộc tập X bằng với số lượng đỉnh thuộc tập Y trong T':

$$\left|V(T') \cap X\right| = \left|V(T') \cap Y\right|.$$

Ta sẽ chứng minh điều này bằng cách phản chứng, giả sử ngược lại, tức là:

$$\left|V(T') \cap X\right| \neq \left|V(T') \cap Y\right|$$

Tức là trong T', số lượng token ban đầu không bằng số lượng vị trí đích. Vì T' là một thành phần liên thông yếu của đồ thị con, nếu số lượng token không cân bằng thì phải có một dòng chảy token không hợp lệ đi vào hoặc đi ra khỏi T', nghĩa là tồn tại một cung e' kết nối T' với phần còn lại của T.

Vì T' chỉ bao gồm các cung có $w(e'; X, Y) > 0$, nếu tồn tại một cung e' kết nối T' với phần còn lại của T, thì cung này cũng phải thỏa mãn $w(e'; X, Y) > 0$. Tuy nhiên, điều này mâu thuẫn với định nghĩa của T', vì ta đã giả sử rằng tất cả các cung thỏa mãn $w(e'; X, Y) > 0$ đều thuộc T'. Do đó, không thể tồn tại một cung đi vào hoặc đi ra khỏi T'. Mâu thuẫn ở trên cho thấy giả thiết $|V(T') \cap X| \neq |V(T') \cap Y|$ là sai. Vì vậy, ta có:

$$|V(T') \cap X| = |V(T') \cap Y|$$

Tiếp theo ta chọn một đỉnh v trong cây con yếu liên thông T' sao cho:

$$N^-_{T'}(v) = \emptyset.$$

Điều này có nghĩa là đỉnh v không có cung vào trong T', hay nói cách khác, **bậc vào** của v bằng 0 trong cây con T'. Ý nghĩa của việc chọn đỉnh này là nếu một đỉnh không có cung vào, nghĩa là nó có thể là một điểm xuất phát tiềm năng cho một đường đi có hướng trong cây con T'.

Xét tổng tất cả các bậc vào trong cây con yếu liên thông T'. Vì mỗi cung trong T' đóng góp một bậc vào và một bậc ra, nên tổng bậc vào trong T' chính bằng tổng số cạnh của T', tức là:

$$\sum_{v \in V(T')} \deg^-(v) = |V(T')| - 1.$$

Điều này xuất phát từ tính chất của một đa cây có hướng (polytree), trong đó có đúng $|V(T')| - 1$ cung để liên thông tất cả các đỉnh. Nếu mỗi đỉnh trong T' đều có bậc vào ít nhất là 1, thì tổng số bậc vào trong T' sẽ ít nhất là $|V(T')|$, lớn hơn số cung $|V(T')| - 1$. Điều này mâu thuẫn với thực tế rằng tổng bậc vào trong T' chỉ bằng $|V(T')| - 1$. Do đó, phải tồn tại ít nhất một đỉnh v có bậc vào bằng 0 trong cây con T'.

Do sự tồn tại của e^* suy ra $|V(T')| \geq 2$, và vì $v \in T'$ nhưng không có cung vào nên nó phải có ít nhất một cung ra, tức là $N^+_{T'}(v) \neq \emptyset$.

Nếu $v \notin X$, ta phải tìm một đỉnh u thuộc tập lân cận đầu ra của v, tức là $u \in N^+_{T'}(v)$, và chọn u sao cho:

$$|C^+(v, u) \cap X| \geq |C^+(v, u) \cap Y|$$

tức là trong phần con của cây T' chứa u, số đỉnh thuộc tập X không nhỏ hơn số đỉnh thuộc tập Y.

Nếu không tồn tại u thỏa mãn điều kiện trên, nghĩa là với mọi đỉnh u, ta đều có:

$$|C^+(v,u) \cap X| < |C^+(v,u) \cap Y|$$

Nghĩa là số lượng đỉnh thuộc tập X trong nhánh chứa u nhỏ hơn số lượng đỉnh thuộc tập Y trong cùng nhánh. Hệ quả của việc này là mỗi nhánh con xuất phát từ v đều có ít đỉnh trong X hơn so với Y.

Vì v là một đỉnh trong T', nên toàn bộ T' có thể xem là tập hợp các nhánh con xuất phát từ v. Nếu trong mỗi nhánh, số đỉnh thuộc X ít hơn số đỉnh thuộc Y, thì khi gộp lại toàn bộ T' ta có:

$$|V(T') \cap X| < |V(T') \cap Y|$$

Tuy nhiên trước đó ta đã chứng minh rằng $|V(T') \cap X| = |V(T') \cap Y|$, dẫn đến mâu thuẫn về số lượng đỉnh. Mẫu thuẫn này giúp ta khẳng định rằng có ít nhất một đỉnh u thỏa mãn điều kiện $\left|C^+(v,u) \cap X\right| \geq \left|C^+(v,u) \cap Y\right|$. Lại có nếu $v \notin X$, khi xét trọng số của cung (v,u), ta có:

$$w((v,u);X,Y) = \left|C^-(v,u) \cap X\right| - \left|C^-(v,u) \cap Y\right|$$

Do ta đã chọn u sao cho:

$$\left|C^+(v,u) \cap X\right| \geq \left|C^+(v,u) \cap Y\right|,$$

Khi $v \notin X$, toàn bộ token của cây con chứa u phải đến từ X nằm sâu bên trong cây Vì $v \notin X$, nó không có token. Nếu ta nhìn vào nhánh con bắt đầu từ v, tất cả token trong X ở cây con chứa u phải đến từ các đỉnh nằm bên dưới u. Tóm lại nếu $v \notin X$, ta sẽ không thể có token từ v để phân phối vào nhánh chứa u. $\Rightarrow$ Tổng số token trong cây con chứa u chỉ có thể đến từ X nằm bên dưới u.

Khi đó, số lượng token trong cây con chứa u có thể giảm. Giả sử $v \in X$, thì số token trong cây con chứa u ít nhất sẽ là số token trong X trước đó cộng thêm token của v. Nhưng nếu $v \notin X$, ta bỏ mất một token có thể đóng góp vào cây con. Điều này có thể làm giảm tổng số lượng token trong cây con của u. Kéo theo đó, giá trị $w((v,u);X,Y)$ có thể trở nên âm, tức là:

$$w((v,u);X,Y) < 0.$$

Nhưng theo giả thiết của bài toán, mọi trọng số đều phải không âm:

$$w(e; X, Y) \geq 0, \quad \forall e \in A.$$

Do đó, mâu thuẫn xảy ra nếu $v \notin X$.Để tránh mâu thuẫn, ta bắt buộc phải có $v \in X$.

Sau khi đã chứng minh rằng $v \in X$, ta tiếp tục xử lý để xây dựng phép ghép các đường đi có hướng từ X đến Y. Ta đã xác định rằng tồn tại một đỉnh $u \in N^+_{T'}(v)$ sao cho:

$$|C^+(v, u) \cap X| \geq |C^+(v, u) \cap Y|$$

Bước tiếp theo là di chuyển token từ v sang u. Khi đó, tập X được cập nhật thành:

$$X' = X \setminus \{v\} \cup \{u\}$$

Tức là, ta loại bỏ v khỏi tập X và thêm u vào tập X.

Với mọi cung $e' \neq (v, u)$, trọng số không thay đổi:

$$w(e'; X', Y) = w(e'; X, Y), \quad \forall e' \neq (v, u).$$

Với cung (v, u), trọng số giảm đi 1:

$$w((v, u); X', Y) = w((v, u); X, Y) - 1.$$

Do ta đã đảm bảo rằng $w((v, u); X, Y) \geq 1$, ta suy ra:

$$w((v, u); X', Y) = w((v, u); X, Y) - 1 \geq 0.$$

Nghĩa là sau khi cập nhật, điều kiện $w(e; X, Y) \geq 0$ vẫn được bảo toàn với mọi cung. Kh đó tổng giá trị trọng số ban đầu:

$$S = \sum_{e \in A} w(e; X, Y).$$

Sau khi di chuyển token từ v sang u, giá trị trọng số của tất cả các cung trừ (v, u) vẫn giữ nguyên, nhưng giá trị của (v, u) giảm đi 1. Do đó:

$$S' = S - 1.$$

Áp dụng giả thiết quy nạp: Do ta đang quy nạp theo tổng giá trị trọng số S, theo giả thiết quy nạp, khi $S' \leq k$, tồn tại một phép ghép các đường đi có hướng từ X' đến Y, ký hiệu là: $\mathcal{P}'$. Sau đó ta chọn một đường đi P' trong tập $\mathcal{P}'$ sao cho điểm xuất phát của nó là:

$$s(P') = u.$$

Do u là điểm mà token đã được di chuyển đến từ v, ta có thể mở rộng đường đi này bằng cách thêm v vào trước nó. Khi đó, đường đi mới sẽ bắt đầu từ v và sau đó tiếp tục theo đường đi P'. Như vậy, ta đã xây dựng được một phép ghép các đường đi đầy đủ từ X đến Y.

4.1.4 Các hệ quả quan trọng

Như đã được định nghĩa hàm $w(e; I^s, I^t)$ đo lường mức độ mất cân bằng giữa số lượng token và vị trí đích trong cây T. Trong bất kỳ quá trình tái cấu hình nào (reconfiguration sequence), để di chuyển token từ trạng thái ban đầu I^s đến trạng thái đích I^t, ta phải duy trì dòng chảy token sao cho số lượng token đi qua mỗi cung e đúng bằng $w(e; I^s, I^t)$. Vì trọng số này chỉ phụ thuộc vào các tập hợp đầu vào và không thay đổi qua các bước tái cấu hình, nên mọi chuỗi tái cấu hình đều có cùng số bước di chuyển bằng tổng:

$$\sum_{e \in A} w(e; I^s, I^t).$$

Từ đó ta có một hệ quả quan trọng sau:

Hệ quả 4.3. *Nếu bài toán có lời giải (yes-instance), thì số lượng token đi qua mỗi cung e trong mọi chuỗi tái cấu hình đều bằng đúng giá trị trọng số $w(e; I^s, I^t)$. Điều này cũng có nghĩa là mọi chuỗi tái cấu hình hợp lệ đều có cùng độ dài bằng tổng:*

$$\sum_{e \in A} w(e; I^s, I^t).$$

Nếu một cung e có giá trị $w(e; I^s, I^t) < 0$, điều này có nghĩa là số lượng token cần di chuyển theo hướng ngược lại của cung nhiều hơn số lượng token có thể di chuyển thuận. Điều này dẫn đến tình trạng tắc nghẽn (bottleneck) trong quá trình tái cấu hình, vì số lượng token không thể chảy đúng hướng. Do đó, nếu tồn tại ít nhất một cung e có trọng số âm, thì bài toán không thể có lời giải. Điều này tiếp tục dẫn đến một hệ quả quan trọng:

Hệ quả 4.4. *Nếu tồn tại một cung e có $w(e; I^s, I^t) < 0$, thì bài toán không thể tái cấu hình.*

4.2 Các token di chuyển tối đa một lần

Trong một chuỗi tái cấu hình, có thể tồn tại một token di chuyển nhiều nhất một lần. Tức là token đó có thể không di chuyển hoặc di chuyển từ vị trí ban đầu đến một

trong các đỉnh kề theo hướng ra của cung và sau đó giữ nguyên vị trí. Những token này có tác động quan trọng trong bài toán, chúng gây ra một số ngoại lệ không cần thiết và nên loại bỏ chúng ngay từ đầu. Trong phần này, tôi sẽ chứng minh rằng các token này có thể được xác định mà không phụ thuộc vào chuỗi tái cấu hình cụ thể và có thể loại bỏ khỏi đầu vào mà không làm thay đổi tính tái cấu hình của bài toán.

4.2.1 Token không di chuyển

Trước tiên chúng ta sẽ xem xét các token không di chuyển trong bất kỳ chuỗi tái cấu hình này. Tập token này có thể xác định hoàn toàn dựa trên cấu trúc của bài toán (T, I^s, I^t), tức là chúng không phụ thuộc vào bất kỳ chuỗi tái cầu hình nào. Để đơn giản hóa ký hiệu, ta viết $w(e)$ thay cho $w(e; I^s, I^t)$, nghĩa là trọng số của cung e trong đồ thị T với tập đầu vào I^s và tập đích I^t.

Định lý 4.5. *Cho một bài toán tái cấu hình có lời giải (yes-instance) được xác định bởi* (T, I^s, I^t). *Khi đó, trong mọi chuỗi tái cấu hình, tập hợp các đỉnh chứa các token không di chuyển được xác định bởi:*

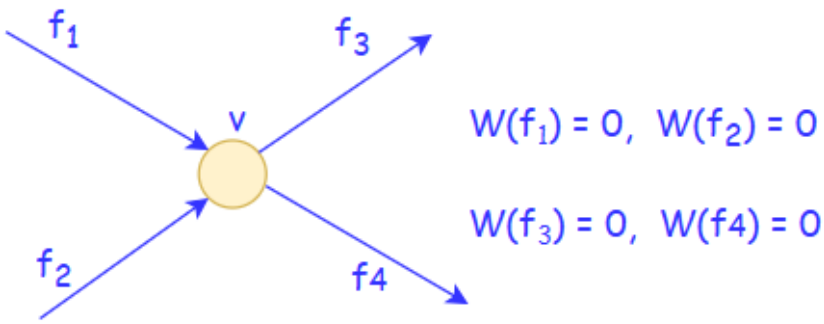
$$R := \{v \in I^s \cap I^t \mid w(e) = 0,\ \forall e \in \Gamma(v)\}$$

Nói cách khác, một đỉnh v thuộc tập R nếu và chỉ nếu:

- v ban đầu có token ($v \in I^s$).
- v cũng thuộc tập đích ($v \in I^t$).
- Mọi cung e kết nối với v đều có trọng số $w(e) = 0$, tức là không có token nào đi qua bất kỳ cung nào liên quan đến v.

Theo **Hệ quả 4.3**, số lượng token đi qua mỗi e trong bất kỳ chuỗi tái cấu hình nào chính là $w(e)$. Nếu một đỉnh v thuộc tập R, thì với mọi cung e kết nối với v, ta có $w(e) = 0, \forall e \in \Gamma(v)$. Điều này có nghĩa là không có token nào di chuyển qua các cung kết nối với v. Vì vậy, token ban đầu đặt trên v không thể di chuyển đến bất kỳ vị trí nào khác và token trên v phải giữ nguyên vị trí ban đầu trong mọi chuỗi tái cấu hình.

Nếu một đỉnh không thuộc R, tức là $v \notin R$, thì tồn tại ít nhất một cung $e \in \Gamma(v)$ sao cho $w(e) > 0$. Điều này có nghĩa là có ít nhất một token phải đi qua e. Trong mọi chuỗi tái cấu hình, token ban đâu đặt tại v phải di chuyển đến một đỉnh kề. Điều này chứng minh rằng nếu $v \notin R$, thì token trên v bắt buộc phải di chuyển.



Hình 4.4: Token cố định

Các token trên đỉnh $v \in R$ sẽ không di chuyển trên bất kỳ chuỗi tái cấu hình nào. Các token này được gọi là Rigid Tokens hay token cố định.

Giả sử có một token cố định tại $v \in R$, nhưng tồn tại một đỉnh kề u sao cho u có một cung e với $w(e) > 0$. Nếu bài toán (T, I^s, I^t) có lời giải, thì một token phải đi qua e trong mọi chuỗi tái cấu hình. Điều này có nghĩa là token tại u sẽ phải di chuyển đến một đỉnh khác tại một thời điểm nào đó. Tuy nhiên, vì token tại v là cố định, nó sẽ không di chuyển, điều này có thể dẫn đễ hai token liền kề nhau sẽ dẫn đến vi phạm đến tính chất tập độc lập của token. Điều này dẫn đến một hệ quả quan trọng:

Hệ quả 4.6. *Nếu một đỉnh $v \in R$ có một đỉnh kề u sao cho u có một cung e với $w(e) > 0$, thì bài toán (T, I^s, I^t) không có lời giải.*

4.2.2 Định nghĩa cung chặn từ những token di chuyển đúng một lần

Sau khi xác định được các token cố định (rigid token), ta tiếp tục xét đến một nhóm token khác đó là các token di chuyển đúng một lần trong mọi chuỗi tái cấu hình.

Một token được gọi là di chuyển đúng một lần nếu nó chỉ thực hiện một bước di chuyển duy nhất từ vị trí ban đầu đến vị trí đích và không tiếp tục di chuyển nữa. Tương tự như các token cố định, ta sẽ chứng minh rằng các token này cũng có thể xác định duy nhất từ dữ liệu đầu vào, tức là không phụ thuộc vào chuỗi tái cấu hình cụ thể.

Trong **Định lý 4.5**, ta đã xác định được tập hợp các token không di chuyển dựa trên điều kiện rằng tất cả các cung kết nối với chúng có trọng số là 0. Bây giờ ta áp dụng lập luận tương tự để tìm tập hợp các token chỉ di chuyển đúng một lần. Các token này phải di chuyển theo một cung e, và chỉ có đúng một cung duy nhất được sử dụng.

Cho một đồ thị có hướng dạng cây $T = (V, A)$ với tập đỉnh V và tập cung A,

cùng với hai tập độc lập ban đầu và đích I^s, I^t. Chúng ta xem xét các token được di chuyển theo một chuỗi tái cấu hình hợp lệ.

Định lý 4.7. *Cho bài toán (T, I^s, I^t) là bài toán có lời giải (yes-instance). Trong mọi chuỗi tái cấu hình, tập hợp các cung được sử dụng bởi các token di chuyển đúng một lần có thể xác định một cách duy nhất từ dữ liệu đầu vào của bài toán, không phụ thuộc vào trình tự di chuyển cụ thể.*

$$B := \{e = (u, v) \in A \mid w(e) = 1, \text{ và } w(e') = 0 \ \forall e' \in (\Gamma(u) \cup \Gamma(v)) \setminus \{e\}\}$$

- $w(e)$ *là số lượng token đi qua cung* e.
- $\Gamma(u)$ *là tập các cung liên thuộc với* u *(cả cung đi vào và đi ra).*
- $\Gamma(v)$ *là tập các cung liên thuộc với* v.

Điều này có nghĩa là một cung $e = u, v \in B$ nếu:

- Có chính xác một token di chuyển qua nó ($w(e) = 1$).
- Không có token nào di chuyển qua bất kỳ cung nào khác kết nối với u hoặc v ($w(e') = 0, \ \forall e' \neq e$).

Chứng minh. Với mọi cung $e = (u, v) \in B$, ta có $u \in I^s \setminus I^t$ và $v \in I^t \setminus I^s$. Với mọi cung $e', \ \forall e' \in (\Gamma(u) \cup \Gamma(v)) \setminus \{e\}\}$, ta có $w(e') = 0$. Từ **Hệ quả 4.4**, số token đi qua mỗi cung chính là trọng số $w(e)$. Vì $w(e) = 1$, điều đó có nghĩa là chỉ có đúng một token duy nhất đi qua cung e. Do không có token nào đi qua các cung khác liên thuộc với u hoặc v, nên token tại u phải di chuyển đến v qua cung e rồi dừng lại. Từ đó ta có phát biểu rằng nếu $e = (u, v) \in B$, token trên u di chuyển đến v và dừng lại.

Ta định nghĩa một tập B' là tập các cung mà token di chuyển chính xác một lần trong ít nhất một chuỗi tái cấu hình. Ta sẽ chứng minh phần đảo của **Định lý 4.7** rằng nếu một cung $e = (u, v)$ không thuộc tập B, thì nó cũng không thể nằm trong tập B', tức là nó không phải là một cung mà token di chuyển đúng một lần trong bất kỳ chuỗi tái cấu hình nào. Giả sử một cung $e = (u, v) \notin B$, thì có hai khả năng:

- $w(e) = 0$.
- Tồn tại một cung khác $e' \neq e$ với $w(e') > 0$ và e' có chung đỉnh với u hoặc v.

Ta sẽ chứng minh rằng trong cả hai trường hợp trên, $e \notin B'$.

Với trường hợp $w(e) = 0$, theo **Hệ quả 4.3**, số lượng token đi qua mỗi cung là duy nhất trong mọi chuỗi tái cấu hình. Do đó, nếu $w(e) = 0$, thì không có token nào đi

qua e trong mọi chuỗi tái cấu hình. Điều này suy ra rằng $e \notin B'$, vì để thuộc B', cung e phải có chính xác một token đi qua trong ít nhất một chuỗi tái cấu hình.

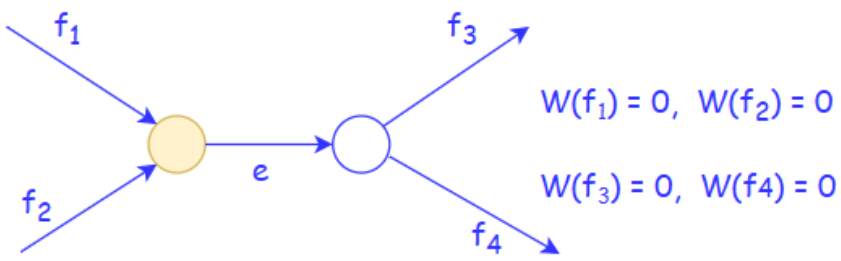
Với trường hợp tồn tại cung $e' \neq e,\ \forall e' \in (\Gamma(u) \cup \Gamma(v)) \setminus \{e\}\}$, tức là có một token đi qua cung e'. Khi đó ta xét các trường hợp:

1. **Nếu $u \notin I^s$, thì không có token nào ở u để di chuyển**
   - Vì $u \notin I^s$, không có token nào khởi đầu tại u
   - Do đó không thể có token nào di chuyển đúng một lần qua e, tức là $e' \notin B'$.
2. **Nếu $u \in I^s$ ta sẽ xét phản chứng**
   Giả sử ngược lại rằng $e = (u, v) \in B'$, tức là tồn tại ít nhất một chuỗi tái cấu hình mà token tại u di chuyển đúng một lần đến v và dừng lại.
   - Do $w(e') > 0$, có ít nhất một token đi qua cung e'
   - Vì token phải bảo toàn số lượng trong quá trình tái cấu hình, điều này có nghĩa là hai token sẽ trở thành liền kề, tức là một token sẽ di chuyển đến vị trí gần token còn lại.
   - Tuy nhiên, điều này mâu thuẫn với nguyên tắc của chuỗi tái cấu hình hợp lệ, vì các token không thể chiếm các vị trí liền kề nếu tập các vị trí phải là một tập độc lập.
   - Do đó, giả thiết rằng $e \in B'$ là sai, suy ra $e \notin B'$.

Ta xét một cung $e = (u, v) \in B$. Theo **Định lý 4.7**, đối với mọi chuỗi tái cấu hình token tại u bắt buộc phải di chuyển đến v theo cung e và sau đó dừng lại. Điều này có nghĩa là u thuộc tập nguồn ban đầu ($u \in I^s$). Đồng thời, v thuộc tập đích cuối cùng($v \in I^t$). Nói cách khác, vị trí ban đầu u và vị trí cuối cùng v cũng phải chứa token trong mọi chuỗi tái cấu hình. Một điều quan trọng cần lưu ý rằng các token phải tạo thành một tập hợp độc lập trong đồ thị trong mọi bước trung gian. Do đó, nếu một token đang nằm ở u và buộc phải di chuyển đến v thì:

- Không có token nào khác có thể đặt ở u sau khi token rời đi.
- Không có token nào khác có thể đặt ở v sau khi token đến.

Điều này có nghĩa là: e sẽ ngăn chặn bất kỳ sự di chuyển nào khác liên quan đến u và v. Do $e \in B$ buộc phải được sử dụng và là con đường duy nhất cho token từ u đến v, nó tạo ra một hàng rào cản trở sự di chuyển của các token khác. Vì lý do này, ta gọi e là một cung chặn (blocking arc) trong đồ thị tái cấu hình



Hình 4.5: Cung chặn

Một minh họa blocking arc từ **Hình 4.5**, nơi token di chuyển duy nhất một lần từ đỉnh nguồn (màu cam) đến đỉnh đích (không màu) qua cung e. Mọi cung khác f_1, f_2, f_3, f_4 đều có trọng số bằng 0, nghĩa là không có token nào đi qua các cung này. Điều này khiến token khi đến đỉnh đích không thể tiếp tục di chuyển, giữ nguyên trạng thái.

4.2.3 Loại bỏ token cố định và cung chặn nhằm đơn giản hóa bài toán

Ta có thể tính toán các rigid tokens (token cố định không di chuyển) và blocking arcs (cung chặn) từ (T, I^s, I^t) trong một thời gian tuyến tính. Gọi R là tập hợp các đỉnh chứa rigid tokens và B là tập hợp chứa các blocking arcs. Xét đồ thị T' thu được từ T bằng cách loại bỏ:

1. Mọi cung f mà một trong hai đầu mút của nó nằm trong R.
2. Mọi cung f không thuộc B nhưng có một đầu mút thuộc một cung trong B.

Sau khi thực hiện các bước loại bỏ trên, mỗi thành phần liên thông trong T' thuộc một trong ba loại sau:

- Một đỉnh cô lập trong R.
- Một thành phần chứa đúng đúng hai đỉnh được kết nối bởi một cung trong B.
- Một thành phần không chứa rigid token hoặc blocking arcs.

Bổ đề sau đây sẽ giúp đơn giản hóa bài toán bằng cách chỉ xét trên T' thay vì T ban đầu.

Bổ đề 4.8. *Giả sử rằng với mọi đỉnh $v \in R$, không tồn tại đỉnh $u \in N(v)$ sao cho đồ thị T có một cung $e \in \Gamma(u)$ thỏa mãn $w(e) > 0$. Khi đó, bài toán có lời giải (yes-instance) đối với (T, I^s, I^t) khi và chỉ khi bài toán cũng có lời giải đối với (T', I^s, I^t).*

Chứng minh chiều xuôi

Với mọi cung $e \in A(T) \setminus A(T')$, ta có $w(e) = 0$. Điều này có nghĩa là không có token nào di chuyển qua cung e tức là việc loại bỏ các cung này không ảnh hưởng đến quá trình tái cấu hình. Do đó, mọi token trong T chỉ di chuyển trong các thành phần còn lại của T'. Ngoài ra, vì mỗi tập độc lập trong T cũng là tập độc lập trong T', nên mọi trạng thái hợp lệ trong T cũng hợp lệ trong T'. Từ đó suy ra, nếu tồn tại một chuỗi tái cấu hình hợp lệ trong T, thì chuỗi đó cũng hợp lệ trong T', chứng minh rằng (T', I^s, I^t) cũng là một *yes-instance*.

Chứng minh chiều ngược

Chúng ta đang chứng minh chiều ngược của **Bổ đề 4.8**: Nếu (T', I^s, I^t) là một *yes-instance*, thì (T, I^s, I^t) cũng là một *yes-instance*.

Giả sử có một chuỗi tái cấu hình hợp lệ từ I^s đến I^t trong T'. Gọi $T_1, T_2, \ldots, T_c$ là các thành phần liên thông yếu của T'. Vì T' không chứa các rigid token hoặc blocking arcs, nên có thể thực hiện tái cấu hình các token trong từng thành thành phần T_i một cách độc lập. Chúng ta có thể giả sử rằng chuỗi tái cấu hình được thực hiện theo từng thành phần liên thông một cách tuần tự:

- **Bước 1**: Xét thành phần liên thông yếu đầu tiên T_1. Các token ban đầu thuộc tập $I^s \cap V(T_1)$ sẽ được di chuyển để đạt trạng thái đích $I^t \cap V(T_1)$.
- **Bước 2**: Sau khi hoàn tất việc tái cấu hình trong T_1, ta tiếp tục với thành phần liên thông yếu thứ hai T_2. Lúc này, các token thuộc tập $I^s \cap V(T_2)$ sẽ di chuyển sao cho chúng khớp với tập $I^t \cap V(T_2)$.
- **Bước 3**: Tiến trình này được tiếp tục cho đến khi tất cả các thành phần $T_1, T_2, \ldots, T_c$ đều đã hoàn thành việc tái cấu hình.

Để biểu diễn quá trình này một cách chính xác, ta ký hiệu chuỗi tái cấu hình của mỗi thành phần T_i là:

$$S_i := \langle I_0^i, I_1^i, \ldots, I_{\ell_i}^i \rangle$$

- I_0^i là trạng thái ban đầu của các token trong T_i, tức là $I_0^i = I^s \cap V(T_i)$.
- $I_{\ell_i}^i$ là trạng thái đích của các token trong T_i, tức là $I_{\ell_i}^i = I^t \cap V(T_i)$.
- Các tập $I_1^i, I_2^i, \ldots, I_{\ell_{i-1}}^i$ là các trạng thái trung gian trong quá trình tái cấu hình của T_i, mỗi bước chỉ di chuyển một token từ một đỉnh sang đỉnh lân cận.

Như vậy, bằng cách thực hiện tái cấu hình theo từng thành phần một cách tuần tự, ta

đảm bảo rằng tiến trình tái cấu hình toàn cục có thể được xây dựng từ các chuỗi tái cấu hình riêng lẻ của từng thành phần T_i.

Bây giờ chúng ta sẽ xây dựng chuỗi tái cấu hình cho toàn bộ đồ thị T, I^s, I^t. Sau khi đã xác định các chuỗi tái cấu hình riêng lẻ cho từng thành phần liên thông yếu của đồ thị T', ta cần ghép các chuỗi này lại thành một chuỗi tái cấu hình hoàn chỉnh. Với hai thành phần liên thông yếu T_i và T_j bất kỳ trong T', ta định nghĩa quan hệ thứ tự giữa chúng như sau:

- T_i đứng trước T_j nếu:
  - T_i chứa một blocking arc duy nhất (x, y) và T_j chứa một đỉnh z là láng giềng của x trong đồ thị T (tức là $z \in N_T(x)$).
  - Ngược lại, nếu T_j chứa một blocking arc duy nhất (y, x) và T_i chứa một đỉnh $z \in N_T(x)$, thì T_j đứng trước T_i

Do đồ thị T là một polytree, nên quan hệ thứ tự giữa các thành phần liên thông yếu là acycic (không có chu trình). Điều này có nghĩa là ta có thể sắp xếp các thành phần $T_1, T_2, ..., T_c$ theo thứ tự sao cho nếu T_j đứng trước T_i, thì chỉ có thể xảy ra khi $j < i$.

Sau khi đã sắp xếp các thành phần theo thứ tự hợp lý, ta xây dựng một chuỗi các tập đỉnh biểu diễn trạng thái của các token trong quá trình tái cấu hình. Gọi chuỗi trạng thái của toàn bộ hệ thống là:

$$\langle I_0, I_1, \ldots, I_\ell \rangle$$

- $I_0 = I^s$ (trạng thái ban đầu).
- $I_\ell = I^t$ (trạng thái đích).

Mỗi bước chỉ di chuyển một token từ một đỉnh u sang đỉnh v, đảm bảo rằng tập đỉnh giữ nguyên tính độc lập trong mọi bước.

- Với mỗi thành phần T_i (đã được sắp xếp theo thứ tự đúng), thực hiện từng bước tái cầu hình trong thành phần này.
- Cho $1 \leq i \leq c,\ 1 \leq j \leq \ell_i$. Với mỗi bước j trong thành phần T_j, trạng thái mới I_{p+j} được xây dựng từ trạng thái trước đó I_{p+j-1} bằng cách di chuyển một token từ đỉnh u sang đỉnh v:

  $$I_{p+j} = I_{p+j-1} \setminus \{u\} \cup \{v\}$$

  với u và v được chọn sao cho:

  $$u \in I^i_{j-1} \setminus I^i_j, \quad v \in I^i_j \setminus I^i_{j-1}.$$

Trong đó, chỉ số p được tính như sau:

$$p = \sum_{1 \le k < i} \ell_k$$

tức là tổng số bước tái cấu hình của tất cả các thành phần T_k trước T_i.

Tiếp theo chúng ta cần chứng minh rằng mỗi tập I_p trong chuỗi tái cấu hình là một tập độc lập trong T. Điều này sẽ đảm bảo rằng tại mỗi bước, các token không vi phạm điều kiện độc lập mỗi khi di chuyển. Để làm được điều này chúng ta sử dụng phương pháp quy nạp trên p, tức là chứng minh cho I_0, sau đó giả sử đúng cho I_{p-1} và chứng minh nó cũng đúng cho I_p.

Xét $0 \le p \le \ell$ tại bước **cơ sở quy nạp** $p = 0$. Ban đầu, tập $I_0 = I^s$, đã là một tập độc lập theo giả thuyết ban đầu nên không cần chứng minh thêm.

Xét $p > 0$, khi chuyển từ trạng thái I_{p-1} sang trạng thái I_p, có đúng một token di chuyển từ một đỉnh u đến một đỉnh v. Với một cung $(u, v) \in A(T)$ ta có:

$$I_{p-1} \setminus I_p = \{u\}, \quad I_p \setminus I_{p-1} = \{v\}$$

Điều này có nghĩa là I_p được tạo ra từ I_{p-1} bằng cách xóa u khỏi I_{p-1} và thêm v vào tập I_{p-1}. Do đó, tập đỉnh gồm u và v phải nằm trong cùng một thành phần liên thông yếu T_j của đồ thị con T':

$$\{u, v\} \subseteq V(T_j), \quad 1 \le j \le c$$

Tiếp theo ta cần chứng minh I_p vẫn là một tập độc lập. Do I_p được xây dựng từ I_{p-1} bằng cách thay thế u bằng v, ta có:

$$I_p \setminus \{v\} = I_{p-1} \setminus \{u\}$$

Điều này có nghĩa là nếu I_{p-1} là một tập độc lập thì $I_p \setminus \{v\}$ cũng là một tập độc lập. Tuy nhiên, để đảm bảo rằng I_p thực sự là một tập độc lập, chúng ta cần kiểm tra xem liệu v có bất kỳ hàng xóm nào trong $I_{p-1} \setminus \{u\}$ hay không. Nếu v không có hàng xóm nào trong tập này, thì I_p vẫn sẽ là một tập độc lập.

Giả sử tồn tại một đỉnh x thuộc $I_{p-1} \setminus \{u\}$, tức là x là một đỉnh có token tại bước trước, mà x lại là hàng xóm của v. Khi đó:

$$x \in I_{p-1} \setminus \{u\}, \quad (x, v) \in A(T)$$

Nhưng theo giả thiết ban đầu, $I_p \cap V(T_j)$ là một tập độc lập trong T_j, do đó không thể có hai đỉnh kề nhau cùng thuộc I_p. Điều này dẫn đến mâu thuẫn.

Do $I_p \cap V(T_j)$ là một tập độc lập trong T_j, điều này có nghĩa là x không thể nằm trong cùng thành phần T_j với v, mà phải thuộc một thành phần $T_{j'}$ khác, tức là:

$$x \in V(T_{j'}), \quad j' \neq j$$

Điều này xảy ra bởi vì x thuộc tập $I_p \setminus \{v\}$ và x không thể cùng xuất hiện với v trong cùng một tập độc lập của T_j. Nhưng nếu x nằm trong một thành phần khác T'_j, thì (x, v) không thể là một cạnh trong đồ thị T, bởi vì các thành phần T_j và T'_j đã được tách rời khi xây dựng T'. Điều này mâu thuẫn với giả định rằng x là hàng xóm của v.

Vì $w((u, v)) > 0$ có nghĩa là có ít nhất một token di chuyển từ u đến v trong quá trình tái cấu hình. Điều này cho thấy rằng u và v không phải là các đỉnh cố định trong tập R tức là $u, v \notin R$.

Trong giả thiết, mọi đỉnh trong R đều không có cạnh đi ra với trọng số $w(e) > 0$. Nếu một đỉnh nằm trọng tập R, tất cả các cạnh đi ra từ đỉnh đó phải có trọng số là 0. Vì x là hàng xóm của v, chúng ta cần kiểm tra xem x có thể thuộc R hay không.

- Nếu $x \in R$, thì tất cả các cung đi ra từ x phải có $w(e) = 0$.
- Nhưng vì x có cạnh nối đến v, nếu $x \in R$, thì $w((x, v)) = 0$.
- Điều này mâu thuẫn với giả thiết rằng $w((u, v)) > 0$, tức là có một token di chuyển dọc theo cung này.

Vậy, $x \notin R$ vì nếu $x \in R$, nó sẽ không có bất kỳ cạnh đi ra nào với trọng số $w(e) > 0$, điều này trái với thực tế rằng nó có cạnh nối với v.

Khi xây dựng cây T' ta đã loại bỏ tất cả các cạnh đi ra từ các đinh trong R và cũng loại bỏ tất cả các cạnh không thuộc tập blocking arc. Do đó nếu (u, v) vẫn còn tồn tại trong T', thì có hai khả năng:

1. (u, v) là một blocking arc
   - Nếu (u, v) là một blocking arc, thì có nghĩa là một khi token di chuyển từ u đến v, nó không thể di chuyển thêm được nữa.
2. x là một đỉnh kết thúc của một blocking arc
   - Nếu x là một đỉnh thuộc endpoint của một blocking arc khác thì nó không thể di chuyển sang vị trí khác do tinh chất của blocking arc.

Nếu (u, v) là một blocking arc, theo quan hệ thứ tự, ta có $j' < j$. Lại có $x \in I^t$ từ cách xây dựng I_p, nghĩa là x thuộc tập đích I^t. $v \in I^t$, theo định nghĩa của blocking arc, do đó token trên u phải di chuyển đến v, và v trở thành một phần của tập đích.

Điều này gây ra mâu thuẫn:

- I^t là tập độc lập, nhưng ở đây, x và v lại cạnh nhau trong đồ thị.
- Điều này vị phạm tính độc lập của I^t.

Suy ra giả thiết ban đầu sai.

Trường hợp x là một endpoint của một blocking arc.

1. Nếu $(x, y) \in B$
   - Theo quan hệ thứ tự $j' < j$, cùng với định nghĩa blocking arc ta suy ra rằng $x \in I^s$.
   - Tuy nhiên, từ cách xây dựng tập I_p, ta biết rằng I_p không chứa bất kỳ đỉnh nào thuộc $I^s \cap V(T_j)$.
   - Điều này mâu thuẫn với giả thiết rằng $x \in I_p$, suy ra giải thiết ban đầu sai.
2. Nếu $(y, x) \in B$
   - Áp dụng quan hệ thứ tự $j < j'$ có nghĩa là thành phần chứa x phải xuất hiện sau T_j
   - Do đó, $x \notin B$, suy ra giả thiết ban đầu sai

Cả hai trường hợp trên đều dẫn đến mâu thuẫn, có nghĩa là giải thiết ban đầu không thể đúng. Do đó, $x \notin I_p$ và **Bổ đề 4.8** được chứng minh.

Dễ dàng nhận thấy rằng nếu T' có nhiều hơn một thành phần liên thông, ta có thể giải quyết vấn đề một cách độc lập trên từng thành phần liên thông. Hơn nữa, bài toán trở nên đơn giản trên một thành phần có kích thước không lớn hơn 2. Vì vậy, ta có thể giả sử rằng polytree đầu vào không chứa bất kỳ đỉnh nào có rigid token hoặc blocking arc.

4.3 Điều kiện cần và đủ tồn tại chuỗi tái cấu hình hợp lệ

Trong mục này, ta sẽ xác định các điều kiện cần và đủ để một **instance** (T, I^s, I^t) của bài toán DTS là một trường hợp có lời giải (*yes-instance*), tức là tồn tại một chuỗi tái cấu hình hợp lệ biến đổi từ tập I^s sang I^t.

4.3.1 Định nghĩa lại tập đường đi và điều kiện kiểm tra ban đầu

Dựa vào **Bổ đề 4.8**, ta có thể giả sử rằng trường hợp đang xét không chứa rigid tokens gồm các token không bao giờ di chuyển trong bất kỳ chuỗi tái cấu hình nào và

blocking arcs là tập các cung chỉ có duy nhất một token đi qua và hai đầu của cung không thể chứa thêm token nào khác.

Giả sử rằng (T, I^s, I^t) là một *yes-instance*, nghĩa là tồn tại một chuỗi tái cấu hình hợp lệ. Khi đó, với mỗi token i, ta xét đường đi có hướng P_i mà token này di chuyển theo trong chuỗi tái cấu hình đó. Khi thu thập tất cả các đường đi đó, ta thu được một tập $P = \{P_1, \ldots, P_k\}$, với $k = |I^s| = |I^t|$. Tập này hiển nhiên là một DPM từ I^s đến I^t, nghĩa là:

- Mỗi token đi từ một đỉnh trong I^s đến một đỉnh trong I^t.
- Các đường đi là disjoint: không chồng lấn nguồn, đích hoặc nội bộ.

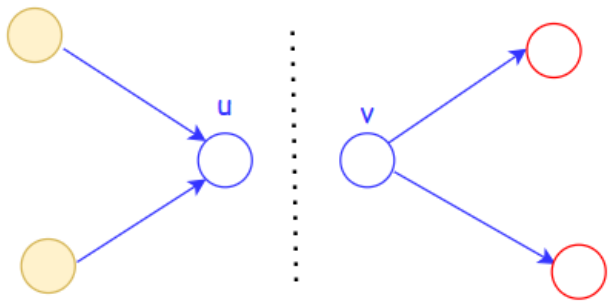
Vì không có token cố định và không có cung chắn, nên độ dài của mỗi đường đi phải thỏa mãn: $|P_i| \geq 2, \quad \forall i \in [k]$.

Tiếp theo, ta cần đảm bảo rằng không có hai token cùng muốn trượt vào cùng một đỉnh trong cùng một thời điểm. Để thấy điều này, giả sử tồn tại hai đường đi P_i, P_j sao cho:

$$s'(P_i) = s'(P_j) = x,$$

tức là bước tiếp theo của cả hai token đều là đỉnh x. Khi đó, cả hai token tại $s(P_i)$ và $s(P_j)$ đều cùng "nhìn về" đỉnh x, dẫn đến xung đột: không thể trượt bất kỳ token nào vào đỉnh x vì nó đang bị "giao tranh". Kết quả là không token nào có thể di chuyển tiếp, chuỗi tái cấu hình không thể thực hiện được. Do đó, cần có điều kiện:

$$s'(P_i) \neq s'(P_j), \quad \forall i \neq j.$$



Hình 4.6: Hai token có cùng successor hoặc cùng predecessor, thì chúng không thể di chuyển

Tương tự, xét điểm cuối (đích đến) của các đường đi. Giả sử hai token i, j cùng kết thúc tại đỉnh y, nhưng lại sử dụng chung một đỉnh ngay trước khi đến y, tức là:

$$t'(P_i) = t'(P_j).$$

Khi đó, việc trượt token vào y sẽ gặp xung đột tương tự như phần nguồn: cả hai token đều muốn đi qua cùng một đỉnh ở bước cuối, dẫn đến va chạm. Vì vậy, ta cần có điều kiện sau để đảm bảo tính hợp lệ:

$$t'(P_i) \neq t'(P_j), \quad \forall i \neq j.$$

Từ các điều kiện đã phân tích ở trên, ta phát biểu bổ đề sau đây nhằm tổng quát hoá tiêu chí cho một *yes-instance*.

Bổ đề 4.9. *Cho một instance (T, I^s, I^t) của bài toán Directed Token Sliding, trong đó không chứa rigid tokens và blocking arcs. Khi đó, instance là một yes-instance khi và chỉ khi tồn tại một tập các đường đi có hướng:*

$$\mathcal{P} = \{P_1, P_2, \ldots, P_k\}, \quad \textit{với } k = |I^s| = |I^t|,$$

thoả mãn tất cả các điều kiện sau (gọi là các điều kiện tập đường đi – path-set conditions):

(P1) Mỗi đường đi có độ dài ít nhất 2: $|P_i| \geq 2, \quad \forall i \in [k]$.

(P2) P là một directed path matching từ I^s đến I^t, tức là:

- *Mỗi token bắt đầu từ một đỉnh duy nhất thuộc I^s và kết thúc tại một đỉnh duy nhất thuộc I^t,*
- *Các đường đi không chồng lấn tại đỉnh nguồn và đỉnh đích.*

(P3) Không có hai token cùng muốn trượt đến một đỉnh giống nhau ngay sau bước đầu tiên:

$$s'(P_i) \neq s'(P_j), \quad \forall i, j \in [k], \quad i \neq j.$$

(P4) Không có hai token cùng sử dụng một đỉnh ngay trước khi đến đích:

$$t'(P_i) \neq t'(P_j), \quad \forall i, j \in [k], \quad i \neq j.$$

Bốn điều kiện (P1)–(P4) được gọi chung là các điều kiện của tập đường đi (path-set conditions). Như vậy, trong phần này, chúng ta đã hoàn tất việc chứng minh điều kiện cần của **Bổ đề 4.9**. Các nội dung còn lại trong **mục 4.3** sẽ tập trung vào việc chứng minh điều kiện đủ, từ đó hoàn thiện toàn bộ lập luận cho bổ đề.

4.3.2 Loại bỏ các cặp đường lệch hướng

Để chứng minh chiều ngược lại của **Bổ đề 4.9** — tức là nếu tồn tại một tập các đường đi có hướng từ I^s đến I^t thỏa mãn các điều kiện (P1)–(P4), thì bài toán là một

yes-instance — ta sẽ tiến hành xây dựng lại chuỗi tái cấu hình phù hợp. Trước tiên, ta sẽ định nghĩa một số khái niệm hỗ trợ:

- Đỉnh trong (*internal vertex*) của một đường đi là một đỉnh nằm trên đường đi nhưng không phải là đỉnh đầu (*source*) hay đỉnh cuối (*sink*).
- Với hai đỉnh u và v trong đồ thị T, ta ký hiệu $\text{dist}(u, v)$ là độ dài của đường đi có hướng duy nhất từ u đến v, nếu tồn tại. Do T là một *polytree* (tức là đồ thị có hướng không chứa chu trình), giữa hai đỉnh bất kỳ sẽ có nhiều nhất một đường đi có hướng.

Gọi P và P' là hai đường đi có hướng từ I^s đến I^t. Ta nói rằng P và P' được gọi là cặp đường đi lệch hướng (biased path pair) nếu thỏa mãn cả hai điều kiện sau:

1. P và P' có một đỉnh chung, ký hiệu là x.
2. Đường đi P bị lệch hơn so với P' về cả phía trước và phía sau đỉnh x, cụ thể:
   - Khoảng cách từ đỉnh đầu của P đến x lớn hơn khoảng cách từ đỉnh đầu của P' đến x:
   $$\text{dist}(s(P), x) > \text{dist}(s(P'), x)$$
   - Khoảng cách từ x đến đỉnh cuối của P cũng lớn hơn khoảng cách từ x đến đỉnh cuối của P':
   $$\text{dist}(x, t(P)) > \text{dist}(x, t(P'))$$

Trước khi tiến hành xây dựng thuật toán, chúng ta cần chứng minh một kết quả quan trọng, giúp đơn giản hóa cấu trúc của các tập đường đi thoả mãn.

Bổ đề 4.10. *Nếu tồn tại một tập các đường đi thỏa mãn các điều kiện đường đi (path-set conditions), thì cũng tồn tại một tập đường đi khác vẫn thỏa mãn các điều kiện đó, nhưng không chứa bất kỳ cặp đường đi lệch (biased path pair) nào.*

Giả sử ta có một tập các đường đi có hướng $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ thoả mãn các điều kiện tập đường đi (P1)–(P4). Với mỗi $i \in [k]$, ta ký hiệu $s_i = s(P_i)$ là đỉnh bắt đầu và $t_i = t(P_i)$ là đỉnh kết thúc của đường đi P_i.

Giả sử $\mathcal{P}$ chứa một cặp đường đi lệch hướng (biased path pair), cụ thể là P_i và P_j, có một đỉnh x chung nằm bên trong (internal vertex) cả hai đường đi, sao cho:

- $\text{dist}(s_i, x) > \text{dist}(s_j, x)$
- $\text{dist}(x, t_i) > \text{dist}(x, t_j)$

Khi đó, ta định nghĩa các đoạn con như sau:

- $P_i[s_i, x]$: đoạn con của P_i từ s_i đến x,
- $P_i[x, t_i]$: đoạn con của P_i từ x đến t_i,
- $P_j[s_j, x]$: đoạn con của P_j từ s_j đến x,
- $P_j[x, t_j]$: đoạn con của P_j từ x đến t_j.

Từ đó, xây dựng hai đường đi mới: $P_i' = P_i[s_i, x] \circ P_j[x, t_j]$ và $P_j' = P_j[s_j, x] \circ P_i[x, t_i]$, trong đó $\circ$ biểu diễn phép nối hai đoạn con.

Đặt $\mathcal{P}' := \mathcal{P} \setminus \{P_i, P_j\} \cup \{P_i', P_j'\}$, tức là thay thế hai đường P_i và P_j bằng hai đường mới P_i' và P_j' trong tập. Ta sẽ chứng minh rằng $\mathcal{P}'$ vẫn thỏa mãn đầy đủ các điều kiện từ P(1) đến P(4):

- **P(2)**: Vì việc hoán đổi không làm thay đổi các đỉnh bắt đầu và kết thúc của đường đi (chỉ thay đổi phần giữa), tập $\mathcal{P}'$ vẫn là một DPM từ I^s đến I^t.
- **(P3) & (P4)**: Do x là đỉnh nội bộ nên:

  $$s'(P_i') = s'(P_i), \quad t'(P_i') = t'(P_j), \quad s'(P_j') = s'(P_j), \quad t'(P_j') = t'(P_i)$$

  Vì theo giả thiết ban đầu, ta có:

  $$\text{(P3)} \quad s'(P_i) \neq s'(P_j),$$
  $$\text{(P4)} \quad t'(P_i) \neq t'(P_j).$$

  Khi hoán đổi đoạn giữa hai đường P_i và P_j, ta chỉ hoán vị cặp khác nhau chứ không tạo ra sự trùng lặp mới nào. Cụ thể:

  $$s'(P_i') = s'(P_i) \neq s'(P_j) = s'(P_j'), \quad t'(P_i') = t'(P_j) \neq t'(P_i) = t'(P_j').$$

  Do đó, hai successor (đỉnh liền sau nguồn) và hai predecessor (đỉnh liền trước đích) trong hai đường đi mới vẫn khác nhau từng cặp, tức là các điều kiện (P3) và (P4) vẫn được bảo toàn.
- **(P1)**: Do tất cả các đoạn con được dùng để tạo thành P_i' và P_j' đều có độ dài dương (vì x là một đỉnh nội bộ), ta có:

  $$|P_i[s_i, x]| > 0, \quad |P_i[x, t_i]| > 0, \quad |P_j[s_j, x]| > 0, \quad |P_j[x, t_j]| > 0.$$

  Do đó, hai đường đi mới thỏa mãn:

  $$|P_i'| = |P_i[s_i, x]| + |P_j[x, t_j]| \geq 2, \quad |P_j'| = |P_j[s_j, x]| + |P_i[x, t_i]| \geq 2.$$

  Suy ra, điều kiện (P1) vẫn được đảm bảo.

Tiếp theo, ta so sánh tổng bình phương độ dài các đường đi trong hai tập P và P'. Ta có:

$$|P_i'| = \text{dist}(s_i, x) + \text{dist}(x, t_j), \quad |P_j'| = \text{dist}(s_j, x) + \text{dist}(x, t_i)$$

Suy ra tổng độ dài hai đường đi mới là:

$$|P_i'| + |P_j'| = \text{dist}(s_i, x) + \text{dist}(x, t_j) + \text{dist}(s_j, x) + \text{dist}(x, t_i)$$

Trong khi tổng độ dài hai đường đi ban đầu là:

$$|P_i| + |P_j| = \text{dist}(s_i, x) + \text{dist}(x, t_i) + \text{dist}(s_j, x) + \text{dist}(x, t_j)$$

Do đó:

$$|P_i'| + |P_j'| = |P_i| + |P_j|$$

Tức là tổng độ dài hai bên là bằng nhau, nhưng vì điều kiện biased yêu cầu:

$$\text{dist}(s_i, x) > \text{dist}(s_j, x), \quad \text{dist}(x, t_i) > \text{dist}(x, t_j)$$

nên việc trao đổi các phần chênh lệch này khiến độ dài riêng rẽ của từng đường bị kéo gần nhau lại tức là độ dài mới của P_i' sẽ nhỏ hơn P_i, và độ dài mới của P_j' sẽ lớn hơn P_j. Lại có bất đẳng thức:

$$a^2 + b^2 > (a - \epsilon)^2 + (b + \epsilon)^2 \quad \text{với } a > b \text{ và } \epsilon > 0.$$

Áp dụng vào $a = |P_i|$, $b = |P_j|$, ta có:

$$|P_i|^2 + |P_j|^2 > |P_i'|^2 + |P_j'|^2$$

Nghĩa là tổng bình phương độ dài của hai đường ban đầu lớn hơn tổng bình phương độ dài của hai đường sau khi tráo đổi. Từ đó sẽ xảy ra hệ quả như sau:

$$\sum_{P \in \mathcal{P}} |P|^2 - \sum_{P' \in \mathcal{P}'} |P'|^2 = |P_i|^2 + |P_j|^2 - |P_i'|^2 - |P_j'|^2 > 0.$$

Do đó, sau mỗi lần hoán đổi một cặp đường đi lệch hướng (biased path pair), tổng bình phương độ dài các đường đi giảm. Vì tổng này là số nguyên không âm, nên quá trình này sẽ dừng lại sau hữu hạn bước. Điều này có nghĩa là bằng cách lặp lại quá trình loại bỏ biased path pairs một cách hữu hạn, ta thu được một tập đường đi mới thỏa mãn đầy đủ các điều kiện (P1)–(P4) và không chứa bất kỳ biased path pair nào.

4.3.3 Xây dựng trình tự di chuyển token không xung đột

Giả sử ta đã có một tập các đường đi $\mathcal{P}^* = \{P_1^*, P_2^*, \ldots, P_k^*\}$ thỏa mãn đầy đủ bốn điều kiện (P1)–(P4) và không chứa bất kỳ cặp *biased path pair* nào. Ta sẽ chứng minh rằng tồn tại một hoán vị $\pi : [k] \to [k]$ sao cho việc di chuyển các token lần lượt theo các đường đi $P^*_{\pi(1)},\ P^*_{\pi(2)},\ \ldots,\ P^*_{\pi(k)}$ là hợp lệ, tức là trình tự di chuyển này không gây ra xung đột và tạo thành một chuỗi tái cấu hình đúng.

Ta nói rằng tập I^s có thể được tái cấu hình thành I^t một cách tuần tự theo từng đường đi trong tập $\mathcal{P}^* = \{P_1^*, P_2^*, \ldots, P_k^*\}$ nếu tồn tại một hoán vị $\pi : [k] \to [k]$ sao cho:

- Token ban đầu tại đỉnh $s(P^*_{\pi(1)}) \in I^s$ sẽ được di chuyển đầu tiên, trượt dọc theo đường đi $P^*_{\pi(1)}$ đến đỉnh $t(P^*_{\pi(1)})$, và giữ nguyên tại đó.
- Tiếp theo, token tại $s(P^*_{\pi(2)})$ cũng được di chuyển tương tự theo đường $P^*_{\pi(2)}$, từ $s(P^*_{\pi(2)})$ đến $t(P^*_{\pi(2)})$, mà không chạm vào bất kỳ token nào đã được di chuyển trước đó.
- Quá trình này tiếp tục cho đến khi tất cả các token trong I^s được di chuyển đến các đỉnh tương ứng trong I^t.

Mỗi bước di chuyển đảm bảo rằng tập các đỉnh chứa token tại mỗi thời điểm luôn là một tập độc lập trong đồ thị T, và không có xung đột (adjacency) nào xảy ra giữa các token đang đứng yên và token đang di chuyển. Kết quả là một chuỗi tái cấu hình hợp lệ (reconfiguration sequence) từ I^s đến I^t, trong đó mỗi token di chuyển dọc theo đường đi định sẵn $P^*_{\pi(i)}$, theo thứ tự $i = 1, 2, \ldots, k$.

Sau khi đã có một tập các đường đi $\mathcal{P}^* = \{P_1^*, \ldots, P_k^*\}$ thỏa mãn điều kiện (P1)-(P4) và không chứa biased path pair, nhiệm vụ còn lại là xác định thứ tự di chuyển token sao cho mỗi bước di chuyển là hợp lệ, không gây xung đột với các token khác. Để làm điều đó, ta cần xác định các quan hệ phụ thuộc giữa các đường đi.

Cho một đỉnh v và một đường đi có hướng P, ta nói rằng v *chạm vào* P (hoặc P *chạm vào* v) nếu:

$$N[v] \cap V(P) \neq \emptyset$$

Nghĩa là đỉnh v có quan hệ láng giềng (kề cạnh hoặc chính nó) với ít nhất một đỉnh nằm trên đường đi P. Khi đó, token tại đỉnh v có thể có khả năng xung đột với token đang hoặc sẽ di chuyển trên đường P.

Xét hai đường đi P_i^* và P_j^* với $i \neq j$, ta định nghĩa hai điều kiện xung đột sau:

(A) Đỉnh đầu (source) của đường P_i^* chạm vào đường P_j^*:

$$s(P_i^*) \textit{ chạm } P_j^*$$

Nghĩa là token tại đỉnh xuất phát của P_i^* có thể gây xung đột với các token đang di chuyển trên P_j^* nếu P_j^* được thực hiện sau P_i^*.

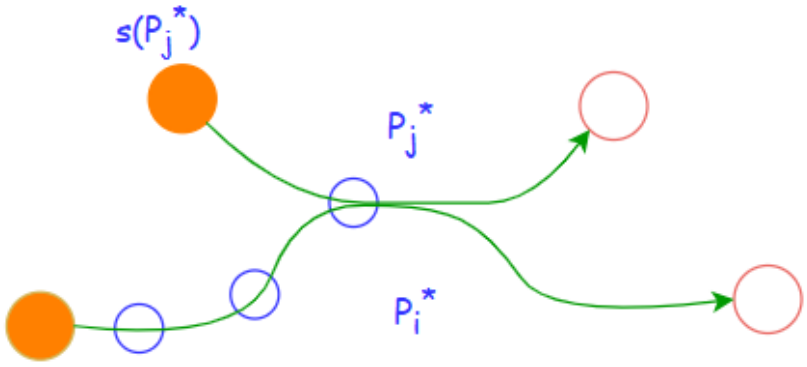
(B) Đỉnh cuối (sink) của đường P_j^* chạm vào đường P_i^*:

$$t(P_j^*) \textit{ chạm } P_i^*$$

Nghĩa là nếu ta di chuyển token trên P_j^* đến đỉnh $t(P_j^*)$, token này có thể gây xung đột với token đang di chuyển trên P_i^* nếu P_j^* được thực hiện trước.

Ta định nghĩa một quan hệ nhị phân $\dashleftarrow$ trên tập chỉ số $[k] = \{1, 2, \ldots, k\}$, tương ứng với các đường đi $P_1^*, P_2^*, \ldots, P_k^*$ trong tập $\mathcal{P}^*$. Với hai chỉ số $i \neq j$, ta nói rằng $i \dashleftarrow j$ khi và chỉ khi một trong 2 điều kiện **(A)** hoặc **(B)** xảy ra. Dựa theo quan hệ $\dashleftarrow$, ta định đồ thị có hướng $G_{\dashleftarrow}$ như sau:

- Tập đỉnh của $G_{\dashleftarrow}$ là tập chỉ số $[k] = \{1, 2, \ldots, k\}$.
- Với 2 chỉ số $i, j \in [k]$, đồ thị có cung từ i đến j khi vào chỉ khi $j \dashleftarrow i$. Nói cách khác: $(i \to j) \in A(G_{\dashleftarrow}) \Leftrightarrow j \dashleftarrow i$.



Hình 4.7: Mối quan hệ $i \dashleftarrow j$ trong đồ thị $G_{\dashleftarrow}$ dựa trên các đường đi P_i^*, P_j^*

Giả sử đồ thị $G_{\dashleftarrow}$ là đồ thị có hướng không chu trình (DAG), khi đó theo định lý cơ bản về DAG, tồn tại ít nhất một đỉnh có bậc ra bằng 0). Gọi đỉnh đó là i, tương ứng với đường đi là P_i^*.

1. **Di chuyển token theo đường P_i^*.**

Đầu tiên ta sẽ kiểm tra điều kiện an toàn để thực hiện việc di chuyển token. Với mọi $j \neq i$, ta có:

- Mọi đỉnh thuộc P_i^* không chạm vào $s(P_j^*) \Leftrightarrow V(P_i^*) \cap N[s(P_j^*)] = \emptyset$. Tức là không có đỉnh nào của P_i^* thuộc vào lân cận của $s(P_j^*)$ hay token đang nằm ở $s(P_j^*)$ sẽ không kề với bất kỳ đỉnh nào trên P_i^*.
- Mọi đỉnh thuộc P_j^* không chạm vào $t(P_i^*) \Leftrightarrow V(P_j^*) \cap N[t(P_i^*)] = \emptyset$. Token của các đường khác sẽ không làm $t(P_i^*)$ kề với chúng.

Điều đó kéo theo hệ quả:

- Token tại $s(P_i^*)$ có thể di chuyển đến $t(P_i^*)$ dọc theo P_i^* mà không tạo thành cặp token kề nhau.
- Sau khi di chuyển, I^s sẽ chuyển thành tập token mới là $I^s \setminus \{s(P_i^*)\} \cup \{t(P_i^*)\}$ vẫn là một tập độc lập.

2. **Cập nhật token và đồ thị** $G_{\leftarrow\!-\!-}$

   Sau khi di chuyển token thứ i ta cập nhật tập độc lập mới sẽ là

   $$I^s \setminus \{s(P_i^*)\} \cup \{t(P_i^*)\}$$

   Sau đó ta sẽ xóa bỏ đỉnh i khỏi đồ thị $G_{\leftarrow\!-\!-}$. Kết quả ta thu được một đồ thị mới

   $$G'_{\leftarrow\!-\!-} = G_{\leftarrow\!-\!-} \setminus \{i\}$$

   Do $G_{\leftarrow\!-\!-}$ là DAG, việc xóa này sẽ không ảnh hưởng đến tính chất này nên suy ra $G'_{\leftarrow\!-\!-}$ vẫn là DAG.

3. **Lặp lại quy trình**

   Ta lặp lại quá trình sau:

   - Ở mỗi bước, chọn một chỉ số i sao cho out-degree$(i) = 0$ trong đồ thị phụ thuộc $G_{\leftarrow\!-\!-}$.
   - Di chuyển token tương ứng theo đường đi P_i^*.
   - Cập nhật tập I^s bằng cách thay thế $s(P_i^*)$ bởi $t(P_i^*)$:

     $$I^s := I^s \setminus \{s(P_i^*)\} \cup \{t(P_i^*)\}$$

   - Xóa đỉnh i khỏi đồ thị phụ thuộc $G_{\leftarrow\!-\!-}$.

   Lặp lại quá trình trên cho đến khi toàn bộ k token đã được di chuyển xong.

Do $G_{\leftarrow\!-\!-}$ là một đồ thị có hướng không chu trình (DAG). Khi đó, tồn tại một *thứ tự tô-pô* của các đỉnh $\{1, 2, \ldots, k\}$, ký hiệu là $\pi(1), \pi(2), \ldots, \pi(k)$, sao cho với mọi cạnh $i \to j$ trong $G_{\leftarrow\!-\!-}$, ta có $\pi^{-1}(i) < \pi^{-1}(j)$. Dựa trên thứ tự tô-pô này, ta lần lượt thực hiện việc di chuyển các token:

- Di chuyển token thứ $\pi(1)$ dọc theo đường đi $P^*_{\pi(1)}$, từ $s(P^*_{\pi(1)})$ đến $t(P^*_{\pi(1)})$.
- Sau đó, di chuyển token thứ $\pi(2)$ dọc theo $P^*_{\pi(2)}$.
- Tiếp tục như vậy cho đến token thứ $\pi(k)$.

Tại mỗi bước, ta đảm bảo rằng trạng thái hiện tại của các token là một tập hợp độc lập, bởi vì các token có khả năng gây xung đột với $P^*_{\pi(i)}$ (theo định nghĩa quan hệ $\dashleftarrow$) đều đã được di chuyển trước đó. Do đó, tồn tại một ánh xạ song ánh $\pi : [k] \to [k]$ thỏa mãn rằng: $I^s \to I^t$ có thể được thực hiện bằng cách di chuyển từng token một theo thứ tự π, mỗi token đi theo đường đi đã định trong tập P^*. Nói cách khác, tập đường đi P^* cho phép một chiến lược tái cấu hình từng bước, tuần tự và không xung đột, do đó thỏa mãn điều kiện đã được đặt ra trước đó.

Việc xác nhận $G_{\dashleftarrow}$ làm một DAG là điều kiện cần để ta có thể thực hiện tái cấu hình tuần tự theo thứ tự tô-pô như đã mô tả trong phần trước (theo ánh xạ π). Nếu tồn tại chu trình trong đồ thị $G_{\dashleftarrow}$ thì sẽ không thể sắp xếp các token theo một thứ tự an toàn để di chuyển mà không gây ra xung đột. Do đó, việc chứng minh bổ đề là tối quan trọng.

Bổ đề 4.11. *Đồ thị $G_{\dashleftarrow}$ là một đồ thị có hướng không chứa chu trình.*

Ta sẽ chứng minh bổ đề bằng phương pháp phản chứng. Giả sử ngược lại rằng đồ thị $G_{\dashleftarrow}$ chứa một chu trình có hướng. Khi đó, tồn tại một dãy các đỉnh tạo thành một chu trình $(1, 2, \ldots, \ell, 1)$ là một chu trình tối thiểu (không chứa chu trình con nhỏ hơn). Để xử lý các chỉ số thuận tiện trong chu trình này, ta định nghĩa các phép toán cộng và trừ theo *modulo* ℓ cụ thể là: $\ell + 1 \equiv 1, \quad 0 \equiv \ell$. Tức là khi chỉ số vượt quá ℓ, ta quay lại đầu chu trình, và khi trừ đi 1 từ 1 thì ta sẽ quay lại đỉnh cuối là ℓ.

Ta xét một chu trình có hướng: $(1, 2, \ldots, \ell, 1)$ trong $G_{\dashleftarrow}$. Giả sử mỗi cặp đường đi liên tiếp (P^*_i, P^*_{i+1}) một đỉnh trong chung (internal vertex) với mọi $i \in [k]$. Ta ký hiệu x và y là đỉnh nguồn và đỉnh đích của đoạn con chung lớn nhất giữa (P^*_i, P^*_{i+1}). Vì T là một polytree nên đoạn đường này là duy nhất. Với mọi $i \in [\ell]$ ta xét từng cung $i \to i + 1$ trong đồ thị $G_{\dashleftarrow}$ tức là có cung có hướng từ đỉnh i đến $i + 1$ suy ra tồn tại mỗi quan hệ $\dashleftarrow$ giữa $i + 1$ và i. Khi đó sẽ phải có ít nhất một trong hai điều kiện **(A)** hoặc **(B)** đúng.

1. **Trường hợp (A):** $s(P^*_i)$ *chạm vào* P^*_{i+1}
   - Khi đó, ta có: $\mathrm{dist}(s(P^*_i), x) \leq 1$ vì x nằm trong đoạn chung giữa P^*_i và P^*_{i+1}.
   - Nếu $\mathrm{dist}(s(P^*_i), x) \geq \mathrm{dist}(s(P^*_{i+1}), x)$ thì xảy ra một trong hai khả năng:

– $s'(P_i^*) = s'(P_{i+1}^*) \Rightarrow$ vi phạm điều kiện (P3) (successor không được trùng).

– $s(P_i^*)$ và $s(P_{i+1}^*)$ kề nhau $\Rightarrow$ vi phạm tính độc lập của tập I^s.

Do đó, bắt buộc phải có:

$$\mathrm{dist}(s(P_i^*), x) < \mathrm{dist}(s(P_{i+1}^*), x)$$

2. **Trường hợp (B):** $t(P_{i+1}^*)$ *chạm vào* P_i^*
   - Tương tự, khi đó: $\mathrm{dist}(y, t(P_{i+1}^*)) \leq 1$
   - Nếu $\mathrm{dist}(y, t(P_i^*)) \leq \mathrm{dist}(y, t(P_{i+1}^*))$ thì sẽ xảy ra một trong hai khả năng:
     – $t'(P_i^*) = t'(P_{i+1}^*) \Rightarrow$ vi phạm điều kiện (P4);
     – $t(P_i^*)$ và $t(P_{i+1}^*)$ kề nhau $\Rightarrow$ vi phạm tính độc lập của tập I^t.

   Vì vậy, phải có:

$$\mathrm{dist}(y, t(P_i^*)) > \mathrm{dist}(y, t(P_{i+1}^*))$$

Lấy một đỉnh z_i bất kỳ trong đoạn chung $x \to y$ (tức là một đỉnh chung trong cả hai đường đi), ta có:

$$\mathrm{dist}(s(P_i^*), z_i) < \mathrm{dist}(s(P_{i+1}^*), z_i) \quad \text{hoặc} \quad \mathrm{dist}(z_i, t(P_i^*)) > \mathrm{dist}(z_i, t(P_{i+1}^*)) \qquad (1)$$

Tức là, theo ít nhất một chiều — từ đỉnh nguồn đến z_i hoặc từ z_i đến đỉnh đích — đường đi P_i^* *lấn át* (dominates) đường đi P_{i+1}^*. Ta đã giả đỉnh rằng tập đường đi P^* không chứa biased path pair, mà cặp (P_i^*, P_{i+1}^*) với z_i như trên thỏa mãn định nghĩa biased path pair. Nên để không mâu thuẫn với giả thiết này, đồng thời vẫn chấp nhận điều kiện (1), ta suy ra rằng:

$$\mathrm{dist}(s(P_i^*), z_i) \leq \mathrm{dist}(s(P_{i+1}^*), z_i) \quad \text{và} \quad \mathrm{dist}(z_i, t(P_i^*)) \geq \mathrm{dist}(z_i, t(P_{i+1}^*)) \qquad (2)$$

Tức là, đường đi P_i^* không ngắn hơn đường đi P_{i+1}^* cả ở phần đầu vào (từ nguồn đến z_i) và phần đầu ra (từ z_i đến đích).

Tiếp theo ta sẽ chứng minh rằng nếu tồn tại một chu trình có hướng trong $G_{\dashleftarrow}$ thì tồn tại một chu trình định hướng trong cây T, điều này sẽ mâu thuẫn với giả thiết rằng T là một polytree. Xét một đồ thị vô hướng nền của T là T^{und}, tức là mọi cung đều được xem như là một cạnh vô hướng. Một walk W sẽ là dãy các đỉnh liên tiếp trong T^{und}. Gọi $\overrightarrow{W}$ là đường đi định hướng (oriented walk) thu được từ W bằng cách gán hướng cho mỗi cạnh trong T^{und}. Nếu cạnh (u, v) cùng hướng với cung trong T, ta gọi đó là *forward*, số lần đi thuận hướng ký hiệu là $\mathrm{fwd}(\overrightarrow{W})$. Nếu ngược hướng, ta gọi là *reverse* và số lần đi ngược hướng sẽ ký hiệu là $\mathrm{rev}(\overrightarrow{W})$.

Xét hai đường đi P_i^*, P_{i+1}^* thỏa mãn điều kiện (2):

$$\mathrm{dist}(s(P_i^*), z_i) \leq \mathrm{dist}(s(P_{i+1}^*), z_i)$$

Gọi z_i là đỉnh chung nội bộ giữa P_i^* và P_{i+1}^*. Ta sẽ dựng được một đường đi:

$$W := s(P_i^*) \to z_i \to s(P_{i+1}^*)$$

Từ đó dựng một đường đi định hướng $\overrightarrow{W}$ trên T theo hướng đi qua z_i. Vì $s(P_i^*)$ **gần** z_i hơn $s(P_{i+1}^*)$, đường đi định hướng $\overrightarrow{W}$ đi ngược hướng nhiều hơn chiều thuận

$$\mathrm{fwd}(\overrightarrow{W}) \leq \mathrm{rev}(\overrightarrow{W})$$

Thậm chí, nếu khoảng cách là nghiêm ngặt thì:

$$\mathrm{dist}(s(P_i^*), z_i) < \mathrm{dist}(s(P_{i+1}^*), z_i) \Rightarrow \mathrm{fwd}(\overrightarrow{W}) < \mathrm{rev}(\overrightarrow{W})$$

Giả sử rằng với mọi $i \in [\ell]$, hai đường đi liên tiếp P_i^* và P_{i+1}^* trong chu trình $(1, 2, \ldots, \ell, 1)$ có một đỉnh chung nội bộ z_i. Như vậy, với mỗi i, ta có:

$$\mathrm{dist}(s(P_i^*), z_i) \leq \mathrm{dist}(s(P_{i+1}^*), z_i)$$

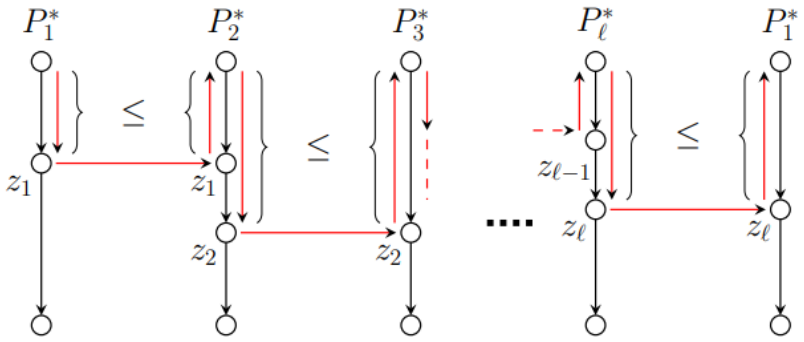
Suy ra, tồn tại một đường đi

$$W_i := s(P_i^*) \to z_i \to s(P_{i+1}^*)$$

trong đồ thị vô hướng T^{und}. Khi hướng các cung theo chiều $s(P_i^*) \to z_i \to s(P_{i+1}^*)$, ta thu được một đường đi định hướng $\overrightarrow{W}_i$ từ $s(P_i^*)$ đến $s(P_{i+1}^*)$. Ghép tất cả các đoạn $\overrightarrow{W}_i$ lại, ta có được một chu trình định hướng khép kín:

$$\overrightarrow{W}_s := s(P_1^*) \to z_1 \to s(P_2^*) \to z_2 \to \cdots \to s(P_\ell^*) \to z_\ell \to s(P_1^*)$$

được minh họa trong **Hình 4.8** dưới đây.



Hình 4.8: Chu trình định hướng khép kín $s(P_1^*) \to z_1 \to s(P_2^*) \to \cdots \to s(P_\ell^*) \to z_\ell \to s(P_1^*)$

Mỗi khung đứng P_i^* trong hình biểu diễn một đường đi từ đỉnh đầu $s(P_i^*)$ đến đỉnh cuối $t(P_i^*)$. Các đỉnh $z_1, z_2, \ldots, z_\ell$ là các đỉnh chung nội bộ giữa hai đường đi liên tiếp P_i^* và P_{i+1}^*, được đặt ở giữa các khung tương ứng. Các mũi tên đỏ thể hiện hướng đi trong chu trình định hướng được xây dựng bằng cách lần lượt nối:

$$s(P_1^*) \to z_1 \to s(P_2^*) \to z_2 \to \cdots \to s(P_\ell^*) \to z_\ell \to s(P_1^*)$$

Chuỗi liên kết này tạo thành một chu trình định hướng khép kín ký hiệu là $\vec{W}_s$, đi qua tất cả các đỉnh bắt đầu $s(P_i^*)$ thông qua các đỉnh chung z_i, là phần trung tâm trong lập luận phản chứng của chứng minh.

Từ chu trình định hướng $\overrightarrow{W}_s$ vừa dựng, ta suy ra rằng tồn tại một cung e trong cây T sao cho:

- Cung e xuất hiện ít nhất một lần trong chu trình $\overrightarrow{W}_s$
- Số lần đi theo chiều thuận của e nhỏ hơn số lần đi theo chiều ngược lại.

Nói cách khác, chu trình định hướng $\overrightarrow{W}_s$ có nhiều bước đi ngược chiều hơn là thuận chiều trên cùng một cung e. Điều này mâu thuẫn với giả thiết rằng T là một polytree. Để tiếp tục chứng minh bằng phản chứng, ta sẽ xét một trường hợp ngược lại: giả sử khoảng cách từ $s(P_1^*)$ đến z_1 không nhỏ hơn khoảng cách từ $s(P_2^*)$ đến z_1, tức là: $\mathrm{dist}(s(P_1^*), z_1) \geq \mathrm{dist}(s(P_2^*), z_1)$. Khi đó, từ phương trình (1), ta có:

$$\mathrm{dist}(z_1, t(P_1^*)) > \mathrm{dist}(z_1, t(P_2^*))$$

Lúc này, ta có thể lập luận tương tự như trên nhưng sử dụng phần cuối của các đường đi $t(P_i^*)$ và dựng một chu trình khép kín $\overrightarrow{W}_t$ bắt đầu từ $t(P_i^*)$ đi ngược lên z_1, đến $t(P_2^*)$, rồi lặp lại chuỗi tương tự cho các đỉnh $t(P_i^*)$ còn lại, kết thúc quay trở lại $t(P_1^*)$. Vì $\mathrm{dist}(z_1, t(P_1^*)) > \mathrm{dist}(z_1, t(P_2^*))$, nên chu trình định hướng $\overrightarrow{W}_t$ cũng sẽ có số lần đi ngược chiều lớn hơn số lần đi thuận chiều, tức là:

$$\mathrm{fwd}(\overrightarrow{W}_t) < \mathrm{rev}(\overrightarrow{W}_t).$$

Một lần nữa, điều này mâu thuẫn với cấu trúc của T là một polytree. Từ hai khả năng dẫn đến mâu thuẫn ta có thể kết luận rằng giả thiết ban đầu rằng $G_{\dashleftarrow}$ chứa chu trình là sai. Do đó $G_{\dashleftarrow}$ là một DAG và và P_i^* và P_{i+1}^* luôn có một đỉnh nội bộ chung (common internal vertex) với mọi $i \in [k]$.

Ta cần chứng minh phần còn lại của **Bổ đề 3.11**: Dưới giả thiết rằng đồ thị $G_{\dashleftarrow}$ có một chu trình định hướng, thì mỗi cặp đường đi liên tiếp P_i^*, P_{i+1}^* trong chu trình

đều phải có một đỉnh nội bộ chung. Giả sử ngược lại tồn tại chỉ số i sao cho P_i, P^*_{i+1} không có đỉnh nội bộ chung. Xem xét trường hợp quan sát được trong định nghĩa quan hệ $\dashleftarrow$, giả sử rằng điều kiện **(A)** xảy ra với $i \dashleftarrow i+1$ tức là: P^*_{i+1} chạm $s(P^*_i)$ (không xét đến trường hợp **(B)** vì đối xứng). Vì độ dài của P^*_i ít nhất là 2 (do thỏa mãn điều kiện **(P1)**), nên nó chứa cung $(s'(P^*_i), x^*)$ trong đó $(s'(P^*_i)$ là đỉnh kề ngay sau đỉnh đầu $(s(P^*_i)$.

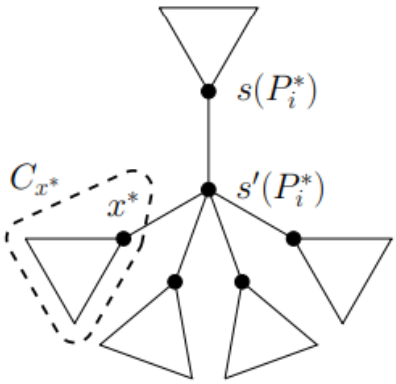
Xét cây T, ta xóa cung $(s'(P^*_{i+1})$ khỏi đồ thị T, và thu được một polyforest (một rừng các cây nhỏ). Gọi C_{x^*} là thành phần liên thông yếu chứa đỉnh x^* trong polyforest này. Ta sẽ chứng minh rằng:

$$V(P^*_{i+1}) \cap C_{x^*} = \emptyset$$

tức là, P^*_{i+1} hoàn toàn nằm ngoài thành phần liên thông chứa x^*. Giả sử tồn tại một đỉnh $v \in V(P^*_{i+1}) \cap C_{x^*}$, thì P^*_{i+1} phải chứa cung $(s'(P^*_i), x^*)$:

- Theo giả thiết: P^*_{i+1} *touches* $s(P^*_i)$,
- Nếu $x^* \in V(P^*_{i+1})$, thì cung $(s'(P^*_i), x^*)$ là cách duy nhất để chạm vào một đỉnh con của $s(P^*_i)$ trong cây T.

Như vậy, $s'(P^*_i) \in V(P^*_{i+1})$, và vì cung $(s'(P^*_i), x^*)$ đi ra khỏi $s'(P^*_i)$, nên ta có $s'(P^*_i) \neq t(P^*_{i+1})$. Ngoài ra, $s(P^*_{i+1}) \neq s'(P^*_i)$ vì I^s là tập độc lập, nên không thể có hai token nằm ở hai đỉnh kề nhau. Ta có thể đưa ra kết luận rằng $s'(P^*_i)$ là một đỉnh nội bộ chung của P^*_i và P^*_{i+1}, điều này mâu thuẫn với giả thiết ban đầu rằng hai đường đi không có đỉnh nội bộ chung. Vậy ta đã chứng minh phản chứng thành công tức là $V(P^*_{i+1}) \cap C_{x^*} = \emptyset$.



Hình 4.9: Thành phần liên thông C_{x^*} sau khi xóa cung $(s'(P^*_i), x^*)$ khỏi cây T.

Giả sử $i-1 = i+1$ tức là $i+1 \dashleftarrow i$ (điều này hàm ý rằng chu trình có độ dài $\ell = 2$). Khi đó, hoặc là P^*_i chạm vào $s(P^*_{i+1})$, hoặc P^*_{i+1} chạm vào $t(P^*_i)$:

- **Trường hợp 1:** P_i^* chạm vào $s(P_{i+1}^*)$. Khi đó, vì $V(P_{i+1}^*) \cap C_{x^*} = \emptyset$ và I^s là tập độc lập, nên suy ra $s(P_{i+1}^*) \in N_T(s'(P_i^*))$, tức là $s'(P_{i+1}^*) = s'(P_i^*)$. Điều này vi phạm điều kiện (P3), vì hai đường đi P_i^* và P_{i+1}^* có cùng successor.
- **Trường hợp 2:** P_{i+1}^* chạm vào $t(P_i^*)$. Khi đó, vì $V(P_{i+1}^*) \cap C_{x^*} = \emptyset$, nên suy ra $x^* = t(P_i^*)$, và đồng thời $s'(P_i^*) \in V(P_{i+1}^*)$. Hơn nữa, do I^s và I^t là các tập độc lập nên:

  $$s'(P_i^*) \neq s(P_{i+1}^*) \quad \text{và} \quad s'(P_i^*) \neq t(P_{i+1}^*).$$

  Điều này chứng tỏ rằng $s'(P_i^*)$ là một đỉnh nội bộ của P_{i+1}^*, mâu thuẫn với giả thiết rằng hai đường đi P_i^* và P_{i+1}^* không có đỉnh nội bộ chung.

Suy ra, trường hợp $i + 1 \dashleftarrow i$ với $\ell = 2$ là không thể xảy ra.

Giả sử $i - 1 \neq i + 1$, điều này có nghĩa là chu trình có độ dài $\ell \geq 3$. Ta sẽ chứng minh mâu thuẫn trong giả định rằng P_i^* và P_{i+1}^* không có đỉnh nội bộ chung. Ta khẳng định rằng:

$$V(P_{i-1}^*) \cap C_{s(P_i^*)} = \emptyset$$

Giả sử ngược lại, tức là: $V(P_{i-1}^*) \cap C_{s(P_i^*)} \neq \emptyset$. Vì $i - 1 \dashleftarrow i$, khi đó có hai khả năng xảy ra hoặc là $s(P_{i-1}^*)$ chạm vào P_i^* hoặc là $t(P_i^*)$ chạm vào P_{i-1}^*

Giả sử tồn tại đỉnh $v \in V(P_{i-1}^*) \cap C_{s(P_i^*)}$ nghĩa là P_{i-1}^* có một đỉnh nằm trong thành phần liên thông $C_{s(P_i^*)}$, tức là cùng phía với $s(P_i^*)$ sau khi loại bỏ cung $(s'(P_i^*), x^*)$ khỏi đồ thị. Khi đó $i - 1 \dashleftarrow i$ nghĩa là ít nhất một trong hai điều kiện sau xảy ra:

(A) $s(P_{i-1}^*)$ chạm vào P_i^*: Vì $s(P_{i-1}^*)$ kề với một đỉnh của P_i^*, tức tồn tại $u \in V(P_i^*)$ sao cho:

$$(s(P_{i-1}^*), u) \in E(T)$$

Mặt khác, ta đã biết:

$$v \in V(P_{i-1}^*) \cap C_{s(P_i^*)}$$

nên cả $s(P_{i-1}^*)$ và $u \in P_i^*$ đều thuộc thành phần liên thông $C_{s(P_i^*)}$. Do đó, $s(P_i^*)$ nằm trong thành phần chứa $s(P_{i-1}^*)$ và u, nên tồn tại đường đi từ $s(P_i^*)$ đến $s(P_{i-1}^*)$, hoặc ít nhất là: $s(P_i^*) \dashleftarrow u \dashleftarrow s(P_{i-1}^*)$

Kết luận: $s(P_i^*)$ chạm vào $P_{i-1}^* \Rightarrow i \dashleftarrow i - 1$

(B) $t(P_i^*)$ chạm vào P_{i-1}^*:

Tồn tại $u \in V(P_{i-1}^*)$ sao cho:

$$(t(P_i^*), u) \in E(T)$$

Vì P_i^* là đường đi từ $s(P_i^*) \to t(P_i^*)$, nên tồn tại đường đi từ $s(P_i^*) \to u$. Mà $u \in P_{i-1}^*$, suy ra $s(P_i^*)$ chạm tới P_{i-1}^* thông qua u.

Kết luận: $s(P_i^*)$ chạm vào $P_{i-1}^* \Rightarrow i \dashleftarrow i-1$

Trong cả hai trường hợp ta đều suy ra rằng $s(P_i^*)$ chạm vào P_{i-1}^* tức là $i \dashleftarrow i-1$. Do theo định nghĩa của "chạm"là tồn tại đỉnh kề. Vậy nên theo định nghĩa quan hệ $\dashleftarrow$, ta có $i \dashleftarrow i-1$, tức là tồn tại cung $i \to i-1$ trong đồ thị $G_{\dashleftarrow}$. Tuy nhiên chu trình đang xét đã tồn tại cung $i-1 \to i$. Việc xuất hiện thêm cung ngược $i \to i-1$ trong cùng chu trình khiến nó không phải một chu trình thuần chiều. Như vậy sẽ xuất hiện thêm một chu trình $(i-1, i, i-1)$ là một chu trình ngắn hơn chỉ gồm 2 đỉnh, điều này sẽ dẫn đến mâu thuẫn với giả thiết ban đầu rằng $(1, 2, 3, \ldots, \ell)$ là chu trình ngắn nhất trong $G_{\dashleftarrow}$.

Giả sử $s'(P_i^*) \in V(P_{i-1}^*)$ tức là đỉnh kế tiếp của nguồn $s(P_i^*)$ (trên đường đi P_i^*) nằm trong đường đi P_{i-1}^*. Theo định nghĩa của quan hệ $\dashleftarrow$, điều này hàm ý: $i \dashleftarrow i-1$ Vì $s'(P_i^*)$ là đỉnh kề với $s(P_i^*)$ và lại thuộc P_{i-1}^*, nên ta có: $s(P_i^*)$ chạm vào P_{i-1}^*. Trong đồ thị $G_{\dashleftarrow}$, tồn tại cung: $i \to i-1$. Lại có chu trình đang xét là: $(1, 2, \ldots, i-1, i, i+1, \ldots, \ell, 1)$. Chu trình này đã chứa cạnh: $i-1 \to i$. Nếu ta thêm cung ngược lại $i \to i-1$, thì sẽ tạo thành một chu trình nhỏ hơn: $(i-1, i, i-1)$. Điều này mâu thuẫn với giả thiết rằng chu trình: $(1, 2, \ldots, \ell, 1)$ là chu trình ngắn nhất trong đồ thị $G_{\dashleftarrow}$. Vì dẫn tới mâu thuẫn, nên ta kết luận rằng: $s'(P_i^*) \notin V(P_{i-1}^*)$. Ta sẽ tiếp tục phân tích hai trường hợp **(i)** $s'(P_i^*) \in V(P_{i+1}^*)$ và **(ii)** $s'(P_i^*) \notin V(P_{i+1}^*)$.

(i) $s'(P_i^*) \in V(P_{i+1}^*)$. Giả sử đỉnh kế tiếp của nguồn $s(P_i^*)$, ký hiệu là $s'(P_i^*)$ nằm trên đường đi P_{i+1}^*. Tức là hai đường đi P_i^* và P_{i+1}^* chia sẻ một đỉnh chung là $s'(P_i^*)$. Vì ta đã giả định rằng hai đường đi P_i^* và P_{i+1}^* không có đỉnh nội bộ chung, nên $s'(P_i^*)$ phải là đỉnh đầu hoặc đỉnh cuối của P_{i+1}^* Tuy nhiên, vì $s'(P_i^*)$ nằm sau $s(P_i^*)$ trên đường đi P_i^*, nên nó không thể là đỉnh bắt đầu của P_{i+1}^*, tức $s'(P_i^*) \neq s(P_{i+1}^*)$. Từ đó suy ra $t(P_{i+1}^*) = s'(P_i^*)$. Tức là, đỉnh cuối của đường đi P_{i+1}^* chính là đỉnh liền sau $s(P_i^*)$ trên đường đi P_i^*.

Theo giả thiết, chu trình $(1, 2, \ldots, \ell, 1)$ là ngắn nhất, nên giữa các đỉnh không có cạnh phụ thêm nào. Vì vậy, đường đi P_{i-1}^* không chứa bất kỳ đỉnh nào lân cận $s'(P_i^*)$, tức là:

$$V(P_{i-1}^*) \cap N_T[s'(P_i^*)] = \emptyset$$

Ngoài ra, ta cũng đã chỉ ra rằng: $V(P_{i-1}^*) \cap C_{s(P_i^*)} = \emptyset$ Xét đỉnh x^* là đỉnh mà cung $(s'(P_i^*), x^*)$ thuộc đường đi P_i^*. Vì P_i^* có độ dài ≥ 2, nên cung này tồn tại. Do P_{i-1}^*

không giao với thành phần liên thông $C_{s(P_i^*)}$, và cũng không lân cận $s'(P_i^*)$, ta có:

$$V(P_{i-1}^*) \subseteq C_{x^*} \quad \text{và} \quad x^* \notin V(P_{i-1}^*)$$

Do chu trình $1 \to 2 \to \cdots \to \ell \to 1$ tồn tại, ta có chuỗi quan hệ:

$$i+1 \dashleftarrow i+2 \dashleftarrow \cdots \dashleftarrow i-1$$

Do đó, tồn tại một chỉ số $m \notin \{i-1, i+1\}$ sao cho: $x^* \in V(P_m^*)$. Suy ra đường đi P_m^* chạm vào P_{i+1}^*, hay $m \dashleftarrow i+1$. Điều này mâu thuẫn với giả thiết rằng chu trình $1 \to 2 \to \cdots \to \ell \to 1$ là chu trình ngắn nhất trong $G_{\dashleftarrow}$, vì tồn tại một cung phụ $m \to i+1$ không nằm trên chu trình ban đầu.

(ii) $s'(P_i^*) \notin V(P_{i+1}^*)$. Khi đó đỉnh liền sau nguồn $s(P_i^*)$ trên đường đi P_i^* không xuất hiện trong đường đi P_{i+1}^*. Đường đi P_i^* bắt đầu từ đỉnh $s(P_i^*)$ và có cung đầu tiên là $(s(P_i^*),\ s'(P_i^*))$ do $|P_i^*| \geq 2$. Do P_{i+1}^* chạm vào $s(P_i^*)$ nhưng không chứa $s'(P_i^*)$, nên:

$$\exists\, v \in N[s(P_i^*)] \setminus \{s'(P_i^*)\} \quad \text{với} \quad v \in V(P_{i+1}^*)$$

Tức là, đường đi P_{i+1}^* phải chứa một đỉnh lân cận của $s(P_i^*)$, nhưng không phải là $s'(P_i^*)$. Do đó, P_{i+1}^* không đi qua cung $(s(P_i^*),\ s'(P_i^*))$ và cũng không đi qua phần của đường P_i^* chứa $s'(P_i^*)$. Từ đó suy ra toàn bộ các đỉnh của P_{i+1}^* nằm trong thành phần liên thông còn lại khi xóa cung $(s(P_i^*),\ s'(P_i^*))$ khỏi đồ thị. Gọi thành phần liên thông còn lại này là $C_{s(P_i^*)}$. Suy ra $V(P_{i+1}^*) \subseteq C_{s(P_i^*)}$.

Xét chuỗi quan hệ $i+1 \dashleftarrow i+2 \dashleftarrow \cdots \dashleftarrow i-1$. Do đây là một chu trình, nên tồn tại chỉ số $m \notin \{i-1, i+1\}$ sao cho:

$$s'(P_i^*) \in V(P_m^*)$$

Tức là, đỉnh $s'(P_i^*)$, mặc dù không thuộc P_{i+1}^* nhưng lại nằm trong thành phần liên thông bị cắt khỏi P_{i+1}^* khi loại bỏ cung $(s(P_i^*),\ s'(P_i^*))$, lại xuất hiện trong một đường đi khác P_m^*. Điều này đồng nghĩa với việc đường đi P_i^* (vì chứa $s'(P_i^*)$) chạm vào đường đi P_m^*, hay nói cách khác $i \dashleftarrow m$. Điều này mâu thuẫn với giả định rằng chu trình $1 \to 2 \to \cdots \to \ell \to 1$ không rút gọn được, tức là không có thêm cung nào ngoài chu trình đó. Điều này hoàn thành việc chứng minh đồ thị $G_{\dashleftarrow}$ là đồ thị có hướng không có chu trình.

Chương 5

THUẬT TOÁN DIRECTED TOKEN SLIDING TRÊN POLYTREE

Trong chương này, hai thuật toán quan trọng cho bài toán Directed Token Sliding trên đồ thị có dạng polytree được trình bày đó là:

- Một thuật toán kiểm tra khả năng tái cấu hình với độ phức tạp tuyến tính theo số đỉnh: $O(|V|)$.
- Một thuật toán xây dựng chuỗi tái cấu hình (nếu tồn tại), với độ phức tạp bình phương: $O(|V|^2)$.

Hai thuật toán này sẽ cùng nhau hoàn thiện việc chứng minh **Định lý 4.1**, đảm bảo rằng bài toán có thể giải quyết hiệu quả trong trường hợp đặc biệt này.

Cho một tập đỉnh $U \subseteq V$, ta định nghĩa hai loại tập láng giềng có hướng của U như sau:

- **Tập láng giềng đi ra (out-neighbors):**

$$N^+(U) := \left(\bigcup_{u \in U} N^+(u) \right) \setminus U$$

  Tức là tập hợp tất cả các đỉnh có cung đi ra từ $u \in U$, không bao gồm các đỉnh thuộc chính U.

- **Tập láng giềng đi vào (in-neighbors):**

$$N^-(U) := \left(\bigcup_{u \in U} N^-(u) \right) \setminus U$$

  Tức là tập các đỉnh có cung đi vào một đỉnh thuộc U, nhưng bản thân không nằm trong U.

Các khái niệm này được sử dụng để định nghĩa ánh xạ dịch chuyển token hợp lệ trong quá trình tái cấu hình.

Giả sử tồn tại một ánh xạ $f : I^s \to N^+(I^s)$ nghĩa là mỗi token ban đầu được

phép di chuyển tới một láng giềng phía trước trong đồ thị. Ta định nghĩa ảnh của ánh xạ f là:

$$f(I^s) := \{f(x) \mid x \in I^s\}$$

Tập này biểu diễn các vị trí trung gian mà token từ tập I^s sẽ đi tới trong bước đầu của quá trình tái cấu hình.

Tương tự, cho ánh xạ: $g : I^t \to N^-(I^t)$ nghĩa là mỗi token mục tiêu sẽ có một vị trí đến từ một láng giềng phía sau (theo hướng cung của đồ thị). Ta định nghĩa:

$$g(I^t) := \{g(y) \mid y \in I^t\}$$

Đây là tập các đỉnh trung gian mà từ đó token sẽ trượt đến vị trí mục tiêu I^t trong bước cuối của chuỗi tái cấu hình.

Mục tiêu của thuật toán là xây dựng được hai ánh xạ f và g như trên, sao cho tồn tại một tập đường đi có hướng từ $f(I^s)$ đến $g(I^t)$, thỏa mãn điều kiện duy trì tính độc lập và không vi phạm các quy tắc trượt token. Khi hai ánh xạ này thỏa mãn các điều kiện kỹ thuật đã nêu ta có thể xác định rằng I^s có thể tái cấu hình thành I^t, dựa vào đó xây dựng được một chuỗi tái cấu hình cụ thể.

5.1 Thuật toán kiểm tra khả năng tái cấu hình

5.1.1 Điều kiện tương đương với tập đường đi qua ánh xạ trung gian

Giả sử tồn tại một tập các đường đi có hướng $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ thỏa mãn các điều kiện đường đi (path-set conditions). Khi đó, ta có thể trích xuất từ mỗi đường đi P_i hai điểm $s(P_i)$ là đỉnh nguồn (vị trí token ban đầu) và đỉnh $s'(P_i)$ là đỉnh kế của đỉnh nguồn tức là điểm đầu mà token sẽ trượt đến. Tương tự ta sẽ có đỉnh $t(P_i)$ là đỉnh đích và đỉnh $t'(P_i)$ là đỉnh liền trước của đỉnh đích.

Nếu ta thay mỗi đường đi P_i bằng đoạn con nối từ đỉnh liền sau nguồn (successor) $s'(P_i)$ đến đỉnh liền trước đích (predecessor) $t'(P_i)$, ta thu được một tâp các đường đi mới tạo thành một ánh xạ một-một (matching) có hường từ tập $X := \{s'(P_i) \mid i \in [k]\}$ đến $Y := \{t'(P_i) \mid i \in [k]\}$ trong đó tập X bao gồm các *successor* (đỉnh ngay sau bước đầu tiên) của các đường đi P_i, trong khi Y là tập các *predecessor* (đỉnh ngay trước đích) của các đường đi đó.

Từ đó, ta định nghĩa hai ánh xạ như sau:

$$f : I^s \to N^+(I^s), \quad \text{với } f(s(P_i)) := s'(P_i)$$

tức là token tại đỉnh $s(P_i)$ sẽ *trượt* sang đỉnh $s'(P_i)$ — là láng giềng phía trước (out-neighbor) theo hướng của đồ thị.

$$g : I^t \to N^-(I^t), \quad \text{với } g(t(P_i)) := t'(P_i)$$

tức là token kết thúc tại $t(P_i)$ sẽ đến từ đỉnh $t'(P_i)$ — là láng giềng phía sau (in-neighbor) theo hướng của đồ thị. Hai ánh xạ f và g phải thỏa mãn bốn điều kiện:

(C1) f và g là ánh xạ một-một (injective).

$$f(s_1) = f(s_2) \Rightarrow s_1 = s_2, \quad g(t_1) = g(t_2) \Rightarrow t_1 = t_2$$

- Điều kiện này có nghĩa là không có hai đỉnh khác nhau trong I^s cùng được ánh xạ đến một đỉnh duy nhất qua f tương tự với g.
- Điều này đảm bảo rằng không có xung đột khi di chuyển token – mỗi token chỉ chiếm một vị trí duy nhất tại mỗi bước.
- Điều kiện này tương đương với (P3) và (P4) trong bộ điều kiện path-set: các đỉnh liền kề trong hướng đi đều không bị trùng nhau giữa các đường đi khác nhau.

(C2) $(s, f(s)) \in A, \quad \forall s \in I^s$

- Ánh xạ f đảm bảo rằng các bước đầu tiên từ mỗi token từ đỉnh s đi đến một đỉnh kề sau theo một cung có hướng trong đồ thị T.
- Ta có thể hiểu rằng tất cả các bước đầu tiên đều hợp lệ theo hướng của đồ thị.

(C3) $(g(t), t) \in A, \quad \forall t \in I^t$

- Tương tự như (C2), nhưng là bước cuối cùng của mỗi token – đi từ một đỉnh liền trước về đến đỉnh t trong I^t.
- Điều này đảm bảo rằng hướng đi ngược về t là hợp lệ trong đồ thị.

(C4) $w(e;\, f(I^s),\, g(I^t)) \geq 0 \quad \forall e \in A$

Điều kiện (C4), tức với mọi cung e, trọng số $w(e; f(I^s), g(I^t)) \geq 0$, chính là điều kiện cần và đủ để tồn tại một phép ghép (matching) từ $f(I^s)$ đến $g(I^t)$ sao cho việc di chuyển token là hợp lệ. Điều này được đảm bảo bởi **Bổ đề 4.2**, vốn đã chứng minh rằng nếu tồn tại một phép ghép các đường đi có hướng (DPM) từ tập nguồn đến tập đích thì mọi trọng số $w(e)$ phải không âm.

5.1.2 Mệnh đề điều kiện cần và đủ

Giả sử tồn tại hai ánh xạ:

- **Ánh xạ** $f : I^s \to N^+(I^s)$:

Đây là một hàm ánh xạ từ mỗi đỉnh $s \in I^s$ trong tập đỉnh nguồn I^s, đến một đỉnh kề với nó theo hướng ra trong đồ thị có hướng. Trong đó với mỗi đỉnh $s \in I^s$, ta có $f(s) \in N^+(s)$, tức là tồn tại một cung có hướng từ s đến $f(s)$ trong đồ thị $G = (V, A)$. Khi đó, tập ảnh của ánh xạ này là:

$$f(I^s) = \{f(s) \mid s \in I^s\} \subseteq V$$

- **Ánh xạ** $g : I^t \to N^-(I^t)$:
  Đây là một hàm ánh xạ từ mỗi đỉnh $t \in I^t$ trong tập đỉnh đích I^t, đến một đỉnh kề với nó theo hướng vào trong đồ thị có hướng. Tức là, với mỗi đỉnh $t \in I^t$, ta có $g(t) \in N^-(t)$, hay tồn tại một cung có hướng từ $g(t)$ đến t trong đồ thị. Tập ảnh tương ứng là:
  $$g(I^t) = \{g(t) \mid t \in I^t\} \subseteq V$$

Hai ánh xạ được nêu trên thỏa mãn bốn điều kiện từ (C1) đến C(4) đã nêu trước đó. Khi đó ta có thể xây dựng được một tập các đường đi có hướng thỏa mãn các điều kiện các tập đường đi từ P(1) đến P(4) như sau:

1. **Tồn tại tập đường đi trung gian** $\mathcal{P}'$
   Theo **Bổ đề 4.2** và điều kiện trọng số C(4), tồn tại một phép ghép các các đường đi có hướng
   $$\mathcal{P}' = \{P'_1, P'_2, \ldots, P'_k\}$$
   nối từ tập $f(I^s)$ đến $g(I^t)$, sao cho mỗi đỉnh trong $f(I^s)$ và $g(I^t)$ chỉ xuất hiện một lần làm điểm đầu và điểm cuối.
2. **Tạo đường đi từ tập** I^s **đến** I^t
   - Mỗi đường đi $P'_i \in \mathcal{P}'$ có thể coi là một đường đi đã được xác định trong không gian các đỉnh $f(I^s)$ và $g(I^t)$. Tuy nhiên, để chuyển đổi chúng thành các đường đi thực sự từ tập I^s đến I^t, ta cần mở rộng các đường đi này.
   - Đầu tiên, ta cần nối từ một đỉnh trong I^s đến đỉnh đầu của đường đi P'_i. Chúng ta nối từ $f^{-1}(s(P'_i))$, tức là đỉnh nguồn $s(P'_i)$ của đường đi P'_i trong không gian $f(I^s)$, quay ngược lại với ánh xạ f^{-1} để tìm ra đỉnh tương ứng trong I^s. Sau đó, ta nối đỉnh $f^{-1}(s(P'_i) \in I^s$ đến $s(P'_i) \in f(I^s)$.
   - Tiếp theo, ta cần nối từ đỉnh cuối của đường đi P'_i đến một đỉnh trong I^t. Chúng ta sẽ nối từ $t(P'_i)$ của đường đi P'_i trong không gian $g(I^t)$ quay ngược lại với ánh xạ g^{-1} để tìm ảnh tương ứng trong I^t. Sau đó, ta nối đỉnh $t(P'_i) \in g(I^t)$ đến $g^{-1}(t(P'_i)) \in I^t$.

- Sau khi thực hiện hai bước trên cho mỗi đường đi P_i' ta sẽ thu được một đường đi P_i mới, đi từ một đỉnh trong I^s đến một đỉnh trong I^t, đi qua phần lõi là đường đi P_i' đã được mở rộng.

3. **Đảm báo tính hợp lệ của các đường đi**
   - Sau khi mở rộng tất cả các đường đi, ta có được một tập $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ của các đường đi thực sự từ I^s đến I^t
   - Mỗi đường đi P_i có độ dài ít nhất là 2, vì ta đã thêm ít nhất một đỉnh vào đầu và một đỉnh vào cuối của mỗi đường đi.
   - Hơn nữa, mỗi cặp đường điP_i và P_j (với $i \neq j$) sẽ không chia sẻ cùng một đỉnh đầu hoặc đỉnh cuối, tức là

$$s(P_i) \neq s(P_j), t(P_i) \neq t(P_j) \forall i \neq j$$

   Tất cả các điều trên sẽ đảm bảo rằng tập $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ của các đường đi thu được thỏa mãn các điều kiện tập đường đi (path-set conditions), và do đó ta đã thành công trong việc tạo ra một tập các đường đi từ I^s đến I^t

Theo lập luận trên, chúng ta thu được các điều kiện cần và đủ sau cho khả năng tái cấu hình như sau:

Bổ đề 5.1. *Đối với trường hợp (instance) không có rigid tokens (token cố định) và không có blocking arcs (cung chặn), tồn tại một tập các đường đi thỏa mãn các điều kiện tập đường đi (path-set conditions) nếu và chỉ nếu tồn tại hai ánh xạ:*

- $f : I^s \to N^+(I^s)$
- $g : I^t \to N^-(I^t)$

Mệnh đề này đưa ra một tiêu chuẩn cực kỳ quan trọng. Thay vì phải kiểm tra trực tiếp sự tồn tại của tập các đường đi phức tạp từ I^s đến I^t, ta chỉ cần kiểm tra xem có thể xây dựng hai ánh xạ trung gian f và g thỏa mãn các điều kiện đơn giản và dễ kiểm tra hơn hay không

Nếu tồn tại hai ánh xạ f và g thỏa mãn bốn điều kiện (C1)-(C4), ta có thể xây dựng một tập các đường đi từ tập nguồn I^s đến tập đích I^t thỏa mãn các bộ điều kiện tương ứng từ (P1)-(P4). Ngược lại, nếu ta đã có một tập các đường đi thỏa mãn điều kiện đường đi (P1)-(P4), thì ta có thể xây dựng hai ánh xạ f và g bằng cách lấy điểm đầu và điểm cuối của đoạn lõi trong mỗi đường đi là các đỉnh thuộc $f(I^s)$ và $g(I^t)$. Việc này đảm bảo ánh xạ thu được thỏa mãn cả bốn điều kiện (C1)-(C4), từ đó khẳng định khả năng tái cấu hình.

5.1.3 Mô tả và phân tích độ phức tạp thuật toán tìm ánh xạ

Theo các **Bổ đề 4.9 và 5.1**, việc kiểm tra xem một bài toán tái cấu hình tập độc lập có thể thực hiện được hay không có thể rút gọn thành việc kiểm tra sự tồn tại của hai ánh xạ trung gian f và g thỏa mãn bốn điều (C1)-(C4). Nếu tồn tại cặp ánh xạ f, g thỏa mãn các điều kiện trên thì có thể khẳng định rằng bài toán có lời giải (*yes-instance*). Nếu không tồn tại, ta kết luận rằng không thể tái cấu hình từ I^s sang I^t.

Để kiểm tra sự tồn tại của các ánh xạ f và g nói trên, chúng ta đề xuất một giải pháp đó là một thuật toán tham lam có độ phức tạp $O(|V|)$, tức tỷ lệ tuyến tính với số lượng đỉnh trong đồ thị. Các bước được thực hiện như sau:

Bước 1 - Khởi tạo

- Chọn một đỉnh bất kỳ làm gốc của cây định hướng T (cây gốc - rooted tree).
- Gán giá trị khởi tạo là "undefined"nghĩa là chưa xác định (ký hiệu là $\perp$) cho tất cả các ánh xạ:

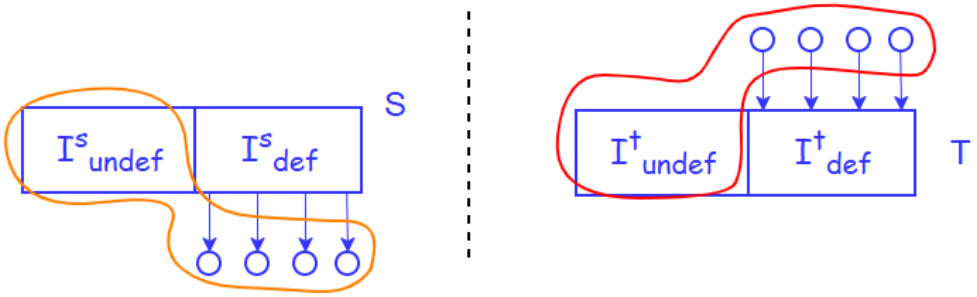
$$f(s) :=\perp \forall s \in I^s; \quad g(t) :=\perp \forall t \in I^t$$

- Từ đó ta định nghĩa bốn tập hỗ trợ như sau:
  - $I^s_{\text{undef}} \subseteq I^s$: tập các đỉnh $s \in I^s$ mà ánh xạ $f(s)$ chưa được xác định;
  - $I^s_{\text{def}} \subseteq I^s$: tập các đỉnh $s \in I^s$ mà ánh xạ $f(s)$ đã được xác định.
  - $I^t_{\text{undef}} \subseteq I^t$ và $I^t_{\text{def}} \subseteq I^t$: được định nghĩa tương tự với ánh xạ g.
- Trong quá trình xây dựng dần các ánh xạ f và g, thuật toán luôn phải duy trì một bất biến quan trọng, nhằm đảm bảo đảm quá trình ánh xạ không làm phá vỡ điều kiện cần và đủ của bài toán.
  - Với mọi đỉnh $s \in I^s_{\text{def}}$ và với mọi $t \in I^t_{\text{def}}$ các điều kiện từ (C1)-(C3) phải được duy trì tức là không có hai đỉnh khác nhau ánh xạ đến cùng một đỉnh (tức ánh xạ là đơn ánh), mỗi cặp $(s, f(s)), (g(t), t)$ là một cung tồn tại trong đồ thị.
  - Với mọi cung $e \in A$, trọng số tương ứng được định nghĩa là:

$$w\left(e;\ I^s_{\text{undef}} \cup f(I^s_{\text{def}}),\ I^t_{\text{undef}} \cup g(I^t_{\text{def}})\right) \geq 0$$

    Điều kiện này đảm bảo rằng: không có cung nào trong đồ thị bị "quá tải" hoặc có xung đột trong quá trình sắp xếp lại các token. Đây là yếu tố cốt lõi giúp đảm bảo rằng việc ghép đường đi sau này là hợp lệ.

Hình 5.1 minh họa trực quan quá trình khởi tạo các tập trạng thái ban đầu trong thuật toán kiểm tra khả năng tái cấu hình *Directed Token Sliding (DTS)*. Cụ thể, với mỗi token xuất phát từ tập I^s và mỗi vị trí đích trong I^t, thuật toán phân chia các đỉnh thành hai nhóm: tập chưa gán ánh xạ (`undef`) và tập đã được gán ánh xạ (`def`).



Hình 5.1: Trạng thái khởi đầu của thuật toán kiểm tra khả năng tái cấu hình DTS bằng cách chia nhỏ tập đỉnh nguồn I^s và đích I^t thành hai nhóm

Ở phía trái hình là tập I^s, trong đó I^s_{undef} gồm các token chưa xác định được đích trung gian $f(s)$, còn I^s_{def} gồm những token đã được ánh xạ đến một đỉnh cụ thể thông qua hàm f. Các mũi tên hướng xuống thể hiện sự gán thành công ánh xạ từ nguồn. Tập mở rộng

$$\tilde{I}^s_{\text{undef}} = I^s_{\text{undef}} \cup f(I^s_{\text{def}})$$

được bao quanh bằng viền cam, dùng làm đầu vào để tính trọng số dòng chảy lệch $w(e)$.

Tương tự, ở phía phải là tập I^t, được chia thành I^t_{undef} (các đỉnh đích chưa được ánh xạ đến bởi token nào) và I^t_{def} (đã được ánh xạ bởi một token thông qua hàm g). Các mũi tên đi xuống thể hiện quá trình ánh xạ thành công $g(t)$. Viền đỏ bao ngoài biểu diễn tập mở rộng

$$\tilde{I}^t_{\text{undef}} = I^t_{\text{undef}} \cup g(I^t_{\text{def}})$$

cũng được dùng để kiểm tra điều kiện $w(e) \geq 0$ trong thuật toán. Việc duy trì rõ ràng hai lớp trạng thái này là nền tảng để thực hiện thuật toán kiểm tra khả năng tái cấu hình một cách chính xác và hiệu quả.

Bước 2 - Lặp

Thuật toán sẽ được lặp lại cho đến khi toàn bộ ánh xạ f và g được xác định, hoặc không thể tiến hành được nữa. Trong mỗi vòng lặp, thuật toán sẽ chọn một đỉnh

v^* chưa được ánh xạ trong tập

$$v^* \in I^s_{\text{undef}} \cup I^t_{\text{undef}}$$

Đỉnh này được chọn sao cho nó có độ sâu lớn nhất trong cây gốc T. Mục đích của việc chọn đỉnh sâu nhất là để tránh việc gán ánh xạ cho những đỉnh "ở giữa"gây ảnh hưởng đến những đỉnh con chưa được xử lý cũng như giúp kiểm soát hướng đi của ánh xạ từ dưới lên, giảm nguy cơ gây mâu thuẫn trong việc giữ bất biến $w(e) > 0$. Đến đây ta sẽ xét 2 trường hợp của v^* từ đó xác định được tập ứng viên ánh xạ.

Trường hợp A : $v^* \in I^s_{\text{undef}}$. Khi đó ta cần tìm các đỉnh $u \in N^+(v^*)$ (tức là các đỉnh kề theo hướng ra từ v^*) để ánh xạ $f(v^*) := u$, sao cho:

- u chưa bị ánh xạ bởi bất kỳ đỉnh nào trước đó hay $u \notin f(I^s_{\text{def}})$.
- Token tại v^* muốn được ánh xạ đến $u \in N^+(v^*)$ qua cung (v^*, u). Khi ánh xạ như vậy sẽ có một đường đi đi qua cung này. Khi đó điều kiện

  $$w\left((v^*, u);\ I^s_{\text{undef}} \cup f(I^s_{\text{def}}),\ I^t_{\text{undef}} \cup g(I^t_{\text{def}})\right) \geq 1$$

  sẽ đảm bảo rằng tổng số token đi qua cung này theo chiều thuận hiện tại chưa vượt quá giới hạn, và vẫn còn chỗ cho ít nhất một token đi thêm qua cung đó.

Tập các đỉnh thỏa mãn điều kiện nêu trên được định nghĩa

$$\text{Ncand}(v^*) := \left\{u \in N^+(v^*) \setminus f(I^s_{\text{def}}) \;\middle|\; w\left((v^*, u);\ I^s_{\text{undef}} \cup f(I^s_{\text{def}}),\ I^t_{\text{undef}} \cup g(I^t_{\text{def}})\right) \geq 1\right\}$$

hay còn được gọi là tập ứng viên có thể gán ánh xạ cho v^*.

Trường hợp B : $v^* \in I^t_{\text{undef}} \setminus I^s_{\text{undef}}$. Khi đó v^* đang chờ được gán ánh xạ $g(v^*)$. Khi đó, ta tìm đỉnh $u \in N^-(v^*)$ (đỉnh kề theo hướng vào) sao cho $u \notin g(I^t_{\text{def}})$ và trọng số cung đủ lớn tương tự với trường hợp A. Khi đó tập các đỉnh thỏa mãn điều kiện nêu trên được định nghĩa

$$\text{Ncand}(v^*) := \left\{u \in N^-(v^*) \setminus g(I^t_{\text{def}}) \;\middle|\; w\left((v^*, u);\ I^s_{\text{undef}} \cup f(I^s_{\text{def}}),\ I^t_{\text{undef}} \cup g(I^t_{\text{def}})\right) \geq 1\right\}$$

Bước 3 - Gán ánh xạ

Sau khi xây dựng được tập ứng viên có thể gán ánh xạ cho v^* ta cần có các bước kiểm tra.

- Nếu $\text{Ncand}(v^*) = \emptyset$. Khi đó, thuật toán dừng lại ngay lập tức và kết luận rằng không tồn tại ánh xạ hợp lệ, tức là bài toán không có lời giải.

- Ncand(v^*) $\neq \emptyset$. Chọn một đỉnh $u^* \in$ Ncand(v^*) có độ sâu lớn nhất (ưu tiên ánh xạ tới đỉnh ở tầng sâu hơn, hạn chế ảnh hưởng tới đỉnh chưa xử lý).
  - Nếu $v^* \in I^s_{\text{undef}}$ ta gán: $f(v^*) := u^*$.
  - Nếu $v^* \in I^t_{\text{undef}}$ ta gán: $g(v^*) := u^*$.

  Sau khi gán ánh xạ, ta cần cập nhật:
  - Di chuyển đỉnh v^* từ tập `undef` sang tập `def` tương ứng: Nếu là ánh xạ $f : I^s_{\text{undef}} \to I^s_{\text{def}}$, ngược lại nếu là ánh xạ $g : I^t_{\text{undef}} \to I^t_{\text{def}}$.
  - Cập nhật lại tập ảnh của ánh xạ: Thêm u^* vào $f(I^s_{\text{def}})$ hoặc $g(I^t_{\text{def}})$.

Tất cả bước trên sẽ được lặp lại cho đến khi tất cả các đỉnh trong I^s và I^t được gán cho một ánh xạ hợp lệ tức là bài toán có lời giải hoặc một đỉnh không có ứng cử viên hợp lệ hay bài toán không có lời giải. Như vậy là kết thức 3 bước cơ bản của thuật toán.

Sau mỗi bước ánh xạ trong vòng lặp, thuật toán phải đảm bảo rằng trọng số của các cung $w(e; X, Y)$ được cập nhật một cách chính xác. Đây là phần quan trọng để duy trì tính bất biến an toàn cho toàn bộ quá trình tái cấu hình tập độc lập. Sau khi mô tả các bước chọn đỉnh, ánh xạ và cập nhật, chúng ta đi đến phần đánh giá hiệu năng và tính chính xác của thuật toán kiểm tra khả năng tái cấu hình. Phần này đặc biệt quan trọng để chứng minh rằng thuật toán chạy trong thời gian tuyến tính.

Trong quá trình lặp của thuật toán, mỗi đỉnh $v \in I^s \cup I^t$ có thể xuất hiện trong vai trò của đỉnh đang xét tối v^* tối đa hai lần. Thật vậy , khi ánh xạ $f(v)$ chưa được xác định, tức là $v \in I^s$ và ban đầu $f(v) = \perp$, thì v thuộc vào tập I^s_{undef}. Trong quá trình thuật toán lặp, nếu v là đỉnh có độ sâu lớn nhất chưa được gán ánh xạ, nó sẽ được chọn làm v^*, và nếu tồn tại đỉnh ứng viên phù hợp, ánh xạ $f(v) := u$ sẽ được gán. Sau khi gán, v được chuyển từ tập I^s_{undef} sang I^s_{def}, và sẽ không còn được chọn lại trong vai trò ánh xạ f. Vì vậy, mỗi đỉnh chỉ được chọn một lần để gán ánh xạ f.

Khi ánh xạ $g(v)$ chưa được xác định, tức là $v \in I^t$ và ban đầu $g(v) = \perp$, thì v thuộc vào tập I^t_{undef}. Nếu trong quá trình thuật toán lặp, v là đỉnh sâu nhất chưa được gán ánh xạ, thì nó sẽ được chọn làm v^*, và ánh xạ $g(v) := u$ sẽ được gán nếu tồn tại một đỉnh ứng viên phù hợp. Sau khi ánh xạ được gán, v được chuyển từ tập I^t_{undef} sang I^t_{def}, và sẽ không còn được chọn lại trong vai trò ánh xạ g. Vì vậy, mỗi đỉnh cũng chỉ được chọn một lần để gán ánh xạ g. Do đó, tổng số vòng lặp trong toàn bộ thuật toán là $O(|I^s| + |I^t|)$, mà $|I^s|, |I^t| \leq |V|$, nên suy ra số vòng lặp tối đa là $2|V|$, tức là giới hạn tuyến tính theo số đỉnh. Điều này góp phần đảm bảo rằng thuật toán có độ phức tạp thời gian tuyến tính.

Nếu với mọi $s \in I^s$ ta có $f(s) \neq \perp$, và với mọi $t \in I^t$ ta có $g(t) \neq \perp$, điều này có nghĩa là tất cả các đỉnh trong tập nguồn và tập đích đều đã được gán ánh xạ hợp lệ. Khi đó, thuật toán kết luận rằng trường hợp đầu vào là một *yes-instance*, tức có thể tái cấu hình từ I^s sang I^t thông qua các bước di chuyển token hợp lệ.

Ngược lại, nếu tồn tại bất kỳ đỉnh nào $s \in I^s$ mà $f(s) = \perp$ hoặc $t \in I^t$ mà $g(t) = \perp$, điều này chứng tỏ không thể tìm được ánh xạ phù hợp cho tất cả token, tức không tồn tại tập đường đi thỏa mãn điều kiện tái cấu hình. Trong trường hợp này, thuật toán kết luận rằng đây là một *no-instance*, nghĩa là không thể thực hiện quá trình tái cấu hình.

Trong mỗi vòng lặp của thuật toán kiểm tra khả năng tái cấu hình, một đỉnh v^* từ $I^s_{\text{undef}} \cup I^t_{\text{undef}}$ được chọn và ánh xạ đến một đỉnh lân cận u^*. Việc ánh xạ này làm một cung duy nhất bị thay đổi trọng số, giảm đi đúng một đơn vị.

Xét trường hợp $v^* \in I^s_{\text{undef}}$. Khi ánh xạ $f(v^*) := u^*$, với $u \in N^+(v^*)$ là đỉnh lân cận theo hướng ra được thực hiện. Việc ánh xạ này chỉ ảnh hưởng đến một cung duy nhất (v^*, u^*) trong khi đó trọng số tất cả các cung khác trong đồ thị sẽ được giữ nguyên. Ta có công thức cập nhật trọng số như sau:

$$
\begin{aligned}
&w\big(e;\, I^s_{\text{undef}} \setminus \{v^*\} \cup f(I^s_{\text{def}}) \cup \{u^*\},\, I^t_{\text{undef}} \cup g(I^t_{\text{def}})\big) \\
&\quad = \begin{cases}
w\big(e;\, I^s_{\text{undef}} \cup f(I^s_{\text{def}}),\, I^t_{\text{undef}} \cup g(I^t_{\text{def}})\big), & \text{nếu } e \neq (v^*, u^*) \\
w\big(e;\, I^s_{\text{undef}} \cup f(I^s_{\text{def}}),\, I^t_{\text{undef}} \cup g(I^t_{\text{def}})\big) - 1, & \text{nếu } e = (v^*, u^*)
\end{cases}
\end{aligned}
$$

Đối với mọi cung $e \neq (v^*, u^*)$ không liên quan đến đỉnh vừa ánh xạ hoặc không nằm trong đường đi thì sẽ không bị ảnh hưởng và trọng số sẽ giữ nguyên. Riêng với cung v^*, u^* là cung mà token mới sẽ sử dụng để di chuyển trọng số sẽ giảm đi đúng 1 đơn vị, biểu thị rằng đã có một token đi qua cung này theo hướng thuận. Việc cập nhật này được thiết kế tối ưu để giữ cho bất biến trọng số $w(e) \geq 0$ luôn được duy trì, đồng thời đảm bảo rằng mỗi lần cập nhật chỉ ảnh hưởng đến đúng một phần tử, giúp thuật toán đạt hiệu năng tuyến tính.

Trong thuật toán, nếu đỉnh v^* thuộc tập I^t nhưng không thuộc I^s, ta cần xác định ánh xạ cho $g(v^*)$. Khi ánh xạ $g(v^*) := u^*$ được thực hiện (với $u^* \in N^-(v^*)$ là một đỉnh kề vào), thì chỉ có duy nhất một cung bị ảnh hưởng, đó là cung (u^*, v^*). Từ đó ta

có công thức cập nhật trọng số như sau:

$$w\big(e;\, I^s_{\text{undef}} \cup f(I^s_{\text{def}}),\, I^t_{\text{undef}} \setminus \{v^*\} \cup g(I^t_{\text{def}}) \cup \{u^*\}\big) = \begin{cases} w\big(e;\, I^s_{\text{undef}} \cup f(I^s_{\text{def}}),\, I^t_{\text{undef}} \cup g(I^t_{\text{def}})\big), & \text{nếu } e \neq (u^*, v^*) \\ w\big(e;\, I^s_{\text{undef}} \cup f(I^s_{\text{def}}),\, I^t_{\text{undef}} \cup g(I^t_{\text{def}})\big) - 1, & \text{nếu } e = (u^*, v^*) \end{cases}$$

Trường hợp này hoàn toàn đối xứng với trường hợp ánh xạ $f(v^*) := u^*$:

Thành phần	**Ánh xạ** f	**Ánh xạ** g
Đỉnh nguồn	$v^* \in I^s_{\text{undef}}$	$v^* \in I^t_{\text{undef}}$
Hướng cung	$(v^*, u^*) \in A$	$(u^*, v^*) \in A$
Cập nhật trọng số cung	Giảm $w((v^*, u^*))$	Giảm $w((u^*, v^*))$
Thành phần bị loại khỏi tập undef	v^* rời khỏi I^s_{undef}	v^* rời khỏi I^t_{undef}
Đỉnh mới thêm vào tập ảnh	$u^* \in f(I^s_{\text{def}})$	$u^* \in g(I^t_{\text{def}})$

Bảng 5.1: So sánh quá trình cập nhật giữa ánh xạ f và g

Do sự đối xứng này, ta gọi cách xử lý này là *symmetric fashion*, nghĩa là không cần viết lại toàn bộ mà chỉ cần đối chiếu logic tương ứng với trường hợp còn lại.Sự đối xứng này giúp thuật toán được thiết kế ngắn gọn và dễ triển khai, đồng thời đảm bảo tính đúng đắn và hiệu quả của việc cập nhật trọng số.

Để đảm bảo hiệu suất xử lý tốt nhất và đạt được độ phức tạp thời gian tuyến tính, thuật toán sử dụng hai bảng (tables) nhằm quản lý hiệu quả thông tin cần thiết trong quá trình ánh xạ:

1. **Bảng trọng số cung**

   Bảng này lưu giữ giá trị hiện tại của hàm trọng số $w(e)$ ứng với mỗi cung $e \in A$ của đồ thị. Việc này cho phép kiểm tra nhanh điều kiện $w(e) \geq 1$ trong quá trình xây dựng tập ứng viên ánh xạ Ncand(v^*). Mỗi lần ánh xạ mới được gán, chỉ một cung duy nhất bị giảm trọng số đi đúng 1 đơn vị, và bảng này sẽ được cập nhật ngay lập tức.

2. **Bảng trạng thái ánh xạ**

   Bảng này quản lý trạng thái ánh xạ hiện tại của từng đỉnh $v \in I^s \cup I^t$. Bảng sẽ cho biết liệu v đã được ánh xạ hay chưa cụ thể đó là $v \in I^s_{\text{def}}$ hay $v \in I^s_{\text{undef}}$ tương tự $v \in I^t_{\text{def}}$ hay $v \in I^t_{\text{undef}}$. Bảng này được cập nhật sau mỗi lần ánh xạ mới được gán, và cho phép kiểm tra nhanh khi cần tìm các đỉnh ứng viên chưa được ánh xạ.

Nhờ duy trì hai bảng này các thao tác trong thuật toán đều được thực hiện với hiệu năng cao.

- Tìm tập ứng viên ánh xạ Ncand(v^*)
  - Được thực hiện bằng cách duyệt qua tập lân cận $N(v^*)$.
  - Thời gian thực hiện là: $O\big(|N(v^*)|\big) = O\big(\deg(v^*)\big)$
  - Do mỗi đỉnh có bậc (degree) hữu hạn, nên tổng chi phí trên toàn bộ thuật toán là tuyến tính.
- **Cập nhật sau mỗi ánh xạ**
  - Mỗi lần ánh xạ chỉ ảnh hưởng đến đúng một cung duy nhất (giảm $w(e)$).
  - Chỉ cần thay đổi đúng một giá trị duy nhất trong bảng.
  - Do đó, chi phí cập nhật là: $O(1)$

Để đánh giá hiệu quả của thuật toán kiểm tra khả năng tái cấu hình, ta phân tích chi tiết các thành phần ảnh hưởng đến thời gian chạy như sau:

1. **Số vòng lặp**

   Mỗi đỉnh $v \in V$ (cụ thể là $v \in I^s \cup I^t$) chỉ có thể được chọn làm đỉnh đang xét v^* tối đa hai lần:
   - Một lần khi ánh xạ $f(v)$ được gán (nếu $v \in I^s$),
   - Một lần khi ánh xạ $g(v)$ được gán (nếu $v \in I^t$).

   Do đó, tổng số vòng lặp của thuật toán là không vượt quá $2|V|$.
2. **Chi phí mỗi vòng lặp**

   Tại mỗi vòng lặp của thuật toán, để xác định được ánh xạ hợp lệ cho đỉnh đang xét v^*, ta cần tìm tập các đỉnh ứng viên ánh xạ Ncand(v^*). Tập này được xác định bằng cách duyệt qua các đỉnh lân cận của v^* trong đồ thị, cụ thể là tập $N(v^*)$. Do đó, thao tác này có độ phức tạp thời gian là $O(|N(v^*)|)$, tức tương đương với $O(\deg(v^*))$, trong đó $\deg(v^*)$ là bậc (số lượng đỉnh kề) của đỉnh v^*.
3. **Tổng chi phí xét các cung**

   Trong quá trình thực thi, mỗi cung $e \in E$ chỉ bị "chạm" tối đa hai lần: một lần khi tra cứu trọng số $w(e)$ để kiểm tra ứng viên ánh xạ Ncand(v^*), và một lần khi cập nhật trọng số sau khi gán ánh xạ $f(v^*) := u^*$ hoặc $g(v^*) := u^*$. Mỗi thao tác chỉ ảnh hưởng đến một cung, nên tổng chi phí xử lý các cung trong toàn bộ thuật toán là $O(|E|)$. Do số cung trong đồ thị là $|E|$, và mỗi cung chỉ bị truy cập hoặc cập nhật tối đa 2 lần, nên tổng chi phí liên quan đến việc xử lý các cung là: $O(|E|)$ Trong trường hợp đặc biệt của bài toán — khi đồ thị là một đa cây có

hướng (polytree) — thì ta có:

$$|E| = |V| - 1 \quad \Rightarrow \quad O(|E|) = O(|V|)$$

Từ tất cả các phân tích trên tổng chi phí sẽ là :

$$\begin{aligned}
T(n) &= \underbrace{O(1) \times 2|V|}_{\text{Cập nhật ánh xạ}} + \underbrace{\sum_{v^* \in V} O(|N(v^*)|)}_{\text{Duyệt lân cận}} \\
&= O(|V|) + \sum_{v \in V} O(\deg(v)) \\
&= O(|V|) + O(|E|) \\
&= O(|V|) + O(|V| - 1) \quad \text{(vì đồ thị là polytree, nên } |E| = |V| - 1) \\
&= O(2|V| - 1) \\
&= \boxed{O(|V|)}
\end{aligned}$$

Tổng chi phí của thuật toán bao gồm các bước cập nhật ánh xạ với chi phí hằng số và duyệt lân cận từng đỉnh. Vì mỗi đỉnh chỉ được xét tối đa hai lần và mỗi cạnh chỉ bị "chạm"tối đa một lần, tổng chi phí qua toàn bộ vòng lặp là

$$O(|V| + |E|).$$

Trong trường hợp đồ thị là polytree, ta có:

$$|E| = |V| - 1 \Rightarrow O(|V| + |E|) = O(|V| + |V| - 1) = \boxed{O(|V|)}.$$

5.1.4 Chứng minh tính đúng đắn của thuật toán

Bổ đề 5.2. *Nếu tồn tại các ánh xạ f và g thỏa mãn các điều kiện trong Bổ đề 4.1, thì thuật toán được mô tả ở trên sẽ xuất ra chính xác một bộ ánh xạ như vậy. Ngược lại, nếu không tồn tại ánh xạ nào thỏa mãn các điều kiện, thuật toán sẽ báo rằng không có ánh xạ hợp lệ.*

Trong suốt quá trình thực hiện, thuật toán luôn duy trì một số điều kiện bất biến (invariants) quan trọng. Cụ thể, tại mỗi thời điểm, các ánh xạ đang được xây dựng là $f : I^s_{\text{def}} \to V$ và $g : I^t_{\text{def}} \to V$ luôn thỏa mãn ba điều kiện đầu tiên (C1)–(C3) được nêu trong **Bổ đề 4.1**, và hàm trọng số $w(e) \geq 0$ được duy trì cho mọi cung $e \in A$. Điều này có nghĩa là, mặc dù thuật toán chưa hoàn tất ánh xạ cho toàn bộ các đỉnh trong I^s và I^t, nhưng phần ánh xạ đã xây dựng luôn hợp lệ tại từng bước. Mỗi lần thuật toán chọn

một đỉnh chưa được ánh xạ (ký hiệu là v^*), nó sẽ cố gắng mở rộng ánh xạ hiện tại bằng cách xác định một ứng viên phù hợp từ tập lân cận. Việc bổ sung ánh xạ mới tại từng bước không phá vỡ khả năng mở rộng toàn cục. Nói cách khác, thuật toán đảm bảo rằng: nếu tồn tại một ánh xạ đúng toàn phần thỏa mãn cả bốn điều kiện (C1)–(C4), thì bất kỳ ánh xạ từng phần nào mà thuật toán đã xây dựng đều có thể tiếp tục được mở rộng một cách an toàn, không tạo ra xung đột hay mâu thuẫn với phần đã được gán trước đó.

Tại mỗi bước của thuật toán, ta xét một đỉnh chưa được ánh xạ, ký hiệu là v^*, và giả sử rằng tập ứng viên khả thi $\text{Ncand}(v^*)$ không rỗng. Tập này chứa các đỉnh mà v^* có thể được ánh xạ tới theo chiều cung, đồng thời không làm vi phạm bất kỳ điều kiện nào về độc lập và trọng số. Giả sử rằng ta đã có một bộ ánh xạ tạm thời f' và g' (tức là ánh xạ đã gán cho một phần các đỉnh), sao cho:

- Với mọi đỉnh $s \in I^s_{\text{def}}$, ánh xạ tạm thời $f'(s) = f(s)$ khớp với ánh xạ đang được xây dựng.
- Với mọi đỉnh $t \in I^t_{\text{def}}$ ánh xạ $g'(t) = g(t)$ cũng tương ứng như vậy.

Giả định thêm rằng ta đang xét trường hợp phía nguồn (tức $v^* \in I^s_{\text{undef}}$), vì trường hợp phía đích (tức $v^* \in I^t_{\text{undef}} \setminus I^s_{\text{undef}}$) là hoàn toàn đối xứng và được xử lý tương tự. Mục tiêu của bước này là chứng minh rằng ta có thể mở rộng ánh xạ hiện tại bằng cách gán $f''(v^*) = u^*$ trong đó u^* là một đỉnh trong $\text{Ncand}(v^*)$, mà vẫn giữ được tính hợp lệ cho toàn bộ ánh xạ mở rộng mới f'' và g''. Khi đó ánh xạ mới sẽ được mở rộng thỏa mãn các điều kiện sau:

- $f''(s) = f(s)$ với mọi đỉnh $s \in I^s_{\text{def}}$ (tức là phần ánh xạ đã được xác định trước đó được giữ nguyên)
- $g''(t) = g(t)$ với mọi đỉnh $t \in I^s_{\text{def}}$
- Và ánh xạ mới được bổ sung là $f''(v^*) = u^*$

Nói cách khác, ta chỉ thêm một ánh xạ mới tại bước này mà không làm ảnh hưởng đến các ánh xạ đã được xây dựng trước đó, và vẫn đảm bảo các điều kiện thuật toán yêu cầu.

Tại bước tiếp theo, ta sẽ xét một đỉnh chưa được ánh xạ $v \in I^s_{\text{undef}}$, và thuật toán đã lựa chọn một đỉnh ứng viên khả thi tương ứng là $u^* \in \text{Ncand}(v^*)$. Đây là đỉnh thỏa mãn các điều kiện để có thể được ánh xạ từ v^* mà không gây vi phạm về độc lập hoặc trọng số. Khi so sánh với ánh xạ tạm thời f' đang được xây dựng, ta nhận thấy có hai

khả năng xảy ra tại đỉnh v^*. **Trường hợp 1** : $f(v^*) = u^*$, tức là ánh xạ tạm thời hiện tại đã trùng với đỉnh ứng viên mà thuật toán chọn. Khi đó ánh xạ này đã hoàn toàn phù hợp. Trong trường hợp này, ta không cần thực hiện bất kỳ thay đổi nào, vì ánh xạ đang xét đã đáp ứng yêu cầu của thuật toán. Do đó, ta có thể giữ nguyên ánh xạ hiện tại và tiếp tục chuyển sang bước xử lý kế tiếp.

Trường hợp 2 : $f'(v) = x^* \neq u^*$, tức là ánh xạ tạm thời khác với đỉnh ứng viên, cần phải xem xét điều chỉnh ánh xạ. Ta sẽ chứng minh rằng việc thay thế ánh xạ hiện tại từ x^* sang u^* là hoàn toàn hợp lệ và cần thiết để duy trì tính đúng đắn. Trước hết do cây định hướng T được gốc hóa (rooted polytree), mọi đỉnh đều có độ sâu được xác định rõ ràng. Ta xây dựng ánh xạ theo thứ tự giảm dần độ sâu, nghĩa là các đỉnh được xử lý từ dưới lên. Vì $x^* \in \text{Ncand}(v^*)$, và theo định nghĩa của $\text{Ncand}(v^*)$, tập ứng viên chỉ chứa nhiều nhất một đỉnh có độ sâu nhỏ hơn v^*, còn lại đều có độ sâu lớn hơn. Từ đó suy ra :

- Đỉnh u^* chắc chắn sâu hơn đỉnh v^* đúng một đơn vị
- Nếu ta không ánh xạ $f(v^*) := u^*$ tại bước này, thì về sau sẽ không còn cơ hội sử dụng cung (v^*, u^*), vì u^* sẽ không còn thuộc tập lân cận chưa xử lý nữa.

Hơn nữa, tại thời điểm này, cung (v^*, u^*) còn đủ trọng số:

$$w\big((v^*, u^*); f'(I^s), g'(I^t)\big) > 0$$

nên việc sử dụng nó là hợp lệ và không gây xung đột. Cuối cùng, do tất cả các ánh xạ f đã được xác định cho các đỉnh trong $N(u^*) \cap I^s \setminus \{v^*\}$, nên $u^* \notin f'(I^s)$ — tức là chưa có đỉnh nào khác sử dụng u^*. Vì vậy, ta có thể định nghĩa ánh xạ mở rộng mới như sau:

$$\begin{aligned}
f''(v^*) &:= u^* \quad \text{— gán ánh xạ mới cho đỉnh đang xét,} \\
f''(v) &:= f'(v) \quad \text{với mọi } v \in I^s \setminus \{v^*\} \quad \text{— giữ nguyên các ánh xạ đã có trước đó,} \\
g'' &:= g' \quad \text{— giữ nguyên ánh xạ đích đang sử dụng.}
\end{aligned}$$

Ánh xạ mới f'', g'' vẫn thỏa mãn toàn bộ các điều kiện trong **Bổ đề 4.1,** và việc mở rộng này đảm bảo thuật toán tiến hành đúng theo lộ trình hợp lệ.

Từ tất cả các lập luận đã trình bày ở trên, ta có thể rút ra kết luận như sau:

- Nếu đầu vào là một *yes-instance* (tức tồn tại một chuỗi tái cấu hình hợp lệ giữa I^s và I^t), thì tại mỗi bước, thuật toán luôn tìm được một ứng viên phù hợp để mở rộng ánh xạ. Khi đó, thuật toán sẽ không dừng giữa chừng, và cuối cùng sẽ xây

dựng được một ánh xạ f, g hoàn chỉnh, thỏa mãn toàn bộ điều kiện của bài toán.

- Ngược lại, nếu đầu vào là một *no-instance* (tức không tồn tại ánh xạ hợp lệ), thì tại một thời điểm nào đó trong quá trình thực thi, thuật toán sẽ không tìm được ứng viên nào thỏa điều kiện. Khi đó, thuật toán sẽ kết thúc sớm, và đưa ra kết luận rằng không tồn tại ánh xạ hợp lệ.

Do đó, thuật toán là đúng đắn hoàn toàn: nó trả về kết quả chính xác cho cả hai trường hợp *yes-instance* và *no-instance*.

5.2 Thuật toán xây dựng chuỗi tái cấu hình

5.2.1 Giả định và định hướng xây dựng chuỗi

Theo **Hệ quả 4.3**, nếu một lời giải là *yes-instance* (tức có thể tái cấu hình từ I^s đến I^t), thì tất cả các chuỗi tái cấu hình hợp lệ đều có cùng độ dài. Điều này cho phép ta chỉ cần tìm một chuỗi duy nhất, thay vì quan tâm đến toàn bộ tập khả dĩ. Trong phần này ta sẽ trình bày một thuật toán có thể xây dựng một chuỗi tái cấu hình hợp lệ với thời gian chạy là $(|V|^2)$, trong đó $|V|$ là số đỉnh của đồ thị.

Bắt đầu bằng việc giả định rằng hai ánh xạ: $f : I^s \to N^+(I^s)$ và $g : I^t \to N^-(I^t)$ đã được xây dựng và thỏa mãn đầy đủ bốn điều kiện (C1)–(C4) như được nêu trong **Bổ đề 4.1**. Trong trường hợp không tồn tại hai ánh xạ như vậy, thì theo bổ đề đó, *instance* được kết luận là không thể tái cấu hình (no-instance). Với giả định rằng f và g đã có, mục tiêu tiếp theo của thuật toán là kết nối các đỉnh trong tập $f(I^s)$ với các đỉnh trong $g(I^t)$ thông qua các đường đi có hướng sao cho không xảy ra xung đột, tức là tránh sự xuất hiện của các *cặp đường đi lệch hướng* (biased path pairs). Trong các bước sau, để đơn giản hóa điều kiện tránh xung đột, ta mở rộng khái niệm cặp đường đi lệch hướng bằng cách cho phép hai đường đi chỉ cần có chung một đỉnh (không nhất thiết phải là đỉnh nội bộ - internal vertex) cũng được coi là gây xung đột — từ đó hình thành khái niệm cặp đường đi lệch hướng yếu (weakly biased path pairs).

Nếu tồn tại một tập đường đi có hướng $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ nối từ tập $f(I^s)$ đến $g(I^t)$, và không chứa bất kỳ cặp đường đi lệch hướng yếu (weakly biased path pair) nào, thì ta hoàn toàn có thể chuyển đổi tập này thành một tập đường đi hoàn chỉnh từ I^s đến I^t thỏa mãn tất cả các điều kiện đã đặt ra trong phần trước. Với mỗi đường đi $P_i \in \mathcal{P}$, ta thực hiện hai thao tác:

- Chèn thêm vào đầu mỗi đường đi: nối từ $f^{-1}(s(P_i)) \in I^s$ đến $s(P_i) \in f(I^s)$,

- Chèn thêm vào cuối mỗi đường đi: nối từ $t(P_i) \in g(I^t)$ đến $g^{-1}(t(P_i)) \in I^t$,

Sau khi thực hiện các thao tác chèn đầu và chèn cuối cho tất cả các $i \in [k]$, ta thu được tập đường đi $\mathcal{P}' = \{P'_1, P'_2, \ldots, P'_k\}$ nối trực tiếp từ I^s đến I^t, đồng thời thỏa mãn toàn bộ điều kiện đường đi (path-set conditions) đã đề ra trước đó (P1–P4). Từ đó nhiệm vụ được đặt ra sẽ là thiết kế một thuật toán xây dựng tập $\mathcal{P}$ từ $f(I^s)$ đến $g(I^t)$ không chứa cặp đường đi lệch hướng yếu, với thời gian chạy tối đa là $O(|V|^2)$.

5.2.2 Định nghĩa độ lệch hướng và cách tính toán

Để xây dựng được tập các đường đi từ $f(I^s)$ đến $g(I^t)$ mà không tạo ra các cặp đường đi lệch hướng yếu, ta cần một công cụ giúp so sánh hướng di chuyển tương đối của các đỉnh trong đồ thị định hướng. Đó là hàm độ lệch hướng, ký hiệu là $d_r(v)$, được định nghĩa như sau: Cho trước một đỉnh gốc tuỳ ý $r \in V$, với mỗi đỉnh $v \in V$, ta xét đường đi duy nhất không có hướng $P_{r,v}$ từ r đến v trong đồ thị cơ sở T^{und} (tức là đồ thị bỏ qua hướng của các cung). Vì T là môt đa cây có hướng, nên giữa r và v luôn tồn tại một đường đi duy nhất trong T^{und}.

Ta chuyển đường đi không hướng $P_{r,v}$ thành một đường đi có hướng $\overrightarrow{P_{r,v}}$ bằng cách định hướng mỗi cạnh theo chiều từ r đến v. Trong đường đi định hướng đó:

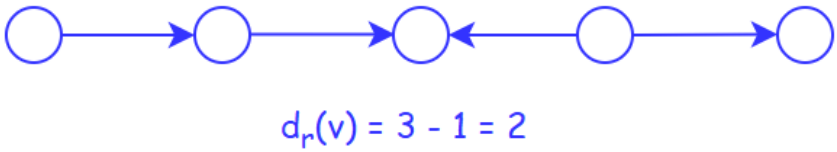
- Gọi $\text{fwd}(\overrightarrow{P_{r,v}})$ là số lượng cung trong T cùng chiều với đường đi.
- Gọi $\text{rev}(\overrightarrow{P_{r,v}})$ là số lượng cung trong T ngược chiều với đường đi.

Khi đó, ta định nghĩa:

$$d_r(v) := \text{fwd}(\overrightarrow{P_{r,v}}) - \text{rev}(\overrightarrow{P_{r,v}})$$

Nếu giá trị $d_r(v)$ lớn, điều đó cho thấy đường đi từ đỉnh gốc r đến v có nhiều cung đi theo đúng chiều định hướng trong đồ thị — tức là việc di chuyển đến v thuận lợi hơn. Ngược lại, nếu $d_r(v)$ nhỏ, nghĩa là đường đi từ r đến v gặp nhiều cung ngược chiều, cho thấy việc di chuyển đến v khó khăn hơn, hoặc không nên ưu tiên chọn v làm đỉnh nguồn trong bước đầu của thuật toán. Sử dụng giá trị $d_r(v)$ giúp thuật toán ưu tiên lựa chọn các cặp đỉnh (nguồn và đích) phù hợp hơn, từ đó giảm khả năng hình thành các đường đi giao nhau theo cách bất lợi (biased), góp phần đảm bảo tính hợp lệ và hiệu quả của toàn bộ chuỗi tái cấu hình.

Một ví dụ trong **Hình 5.2** mô tả một đường đi từ gốc r đến một đỉnh v trong một đa cây có hướng. Có 3 cung đi theo hướng từ trái sang phải (tức là thuận chiều từ gốc đến v). Có 1 cung đi ngược chiều (mũi tên từ phải sang trái) vậy nên $d_r(v) = 3 - 2 = 1$.



Hình 5.2: Minh họa cách tính độ lệch hướng $d_r(v)$

Điều này thể hiện rằng đỉnh v có độ lệch thuận là 2 — nghĩa là số lượng cung thuận chiều vượt trội hơn so với cung ngược, giúp xác định mức độ "gần"với gốc theo chiều hợp lệ trong bài toán Directed Token Sliding.

5.2.3 Mô tả thuật toán xây dựng các đường đi

Dựa trên các giá trị $d_r(v)$ đã được tính cho từng đỉnh $v \in V$. thuật toán sẽ tiến hành xây dựng một cách tuần tự các đường đi có hướng $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$, sao cho mỗi đường đi P_i nối từ một tập $f(I^s)$ đến một đỉnh trong $g(I^t)$, đồng thời tránh được các cặp đường đi lệch hướng yếu.

Tại bước thứ i của thuật toán, ta sẽ xây dựng đường đi P_i. Khi đó ta định nghĩa:

- $X_i \subseteq f(I^s)$: là tập các đỉnh nguồn chưa được sử dụng trong các đường đi $P_1, \ldots, P_{i-1}$.
- $Y_i \subseteq g(I^t)$: là tập các đỉnh đích chưa được sử dụng trong các đường đi đã được tạo ra trước đó.

Trong toàn bộ quá trình xây dựng tập đường đi, thuật toán duy trì một bất biến quan trọng:

$$\forall e \in A, \quad w(e; X_i, Y_i) \geq 0$$

Điều này có nghĩa là, tại thời điểm bước thứ i, trọng số w trên mọi cung e được tính dựa trên các tập đỉnh hiện tại X_i và Y_i, luôn đảm bảo lớn hơn hoặc bằng 0. Theo **Bổ đề 4.2**, điều này đồng nghĩa với việc luôn tồn tại một phép ghép các đường đi hợp lệ từ X_i đến Y_i, tức thuật toán có thể tiếp tục xây dựng thêm một đường đi mà không vi phạm ràng buộc nào.

Tiếp theo, tại từng bước i cụ thể của thuật toán, ta thực hiện việc lựa chọn và xây dựng một đường đi P_i nối một đỉnh từ tập nguồn $f(I^s)$ đến một đỉnh trong tập đích $g(I^t)$. Các bước chi tiết của thuật toán được mô tả như sau:

Bước 1 : Chọn đỉnh nguồn x

Trong tập $X_i \subseteq f(I^s)$, tức là các đỉnh nguồn chưa được sử dụng ở các bước trước đó, ta lựa chọn một đỉnh $x \in X_i$ sao cho giá trị độ lệch hướng $d_r(x)$ là nhỏ nhất trong số các đỉnh còn lại.

- Giá trị $d_r(x)$ đại diện cho mức độ "gần" với gốc r, nghĩa là số lượng cung thuận chiều từ gốc đến x ít hơn so với số lượng cung ngược chiều.
- Việc chọn đỉnh x có $d_r(x)$ nhỏ giúp ưu tiên xử lý trước các đỉnh ở "gần gốc", tránh nguy cơ tạo giao cắt bất lợi (*biased path pairs*) trong các bước sau.

Bước 2 : Xác định tập đích khả thi

Sau khi chọn được đỉnh nguồn x, ta xác định được tập các đỉnh đích khả thi tương ứng bằng cách xét:

$$Y_i' \subseteq Y_i := \{y \in Y_i \mid \exists P_{x \to y} : \forall e \in P_{x \to y},\ w(e; X_i, Y_i) > 0\}$$

Tập Y_i' được định nghĩa là tập con của Y_i bao gồm tất cả các đỉnh $y \in Y_i$ sao cho tồn tại một đường đi có hướng từ đỉnh $x \in X_i$ đến đỉnh y, ký hiệu là $P_{x \to y}$, mà mọi cung e nằm trên đường đi đó đều có trọng số thỏa mãn điều kiện $w(e; X_i, Y_i) > 0$.

- $Y_i \subseteq g(I^t)$: là tập các đỉnh đích chưa được sử dụng trong các đường đi đã được tạo ra trước đó.
- Với mỗi đỉnh $y \in Y_i$, ta kiểm tra xem có tồn tại đường đi có hướng từ $x \to y$ sao cho tất cả các cung e nằm trên đường đi này đều thỏa mãn $w(e; X_i, Y_i) > 0$.
- Điều kiện $w(e; X_i, Y_i) > 0$ đảm bảo rằng việc sử dụng đường đi từ x đến y không gây quá tải trên bất kỳ cung nào trong đồ thị.
- Theo bất biến của thuật toán (xem lại **Bổ đề 4.2**), ta biết rằng luôn tồn tại ít nhất một phép ghép các đường đi, nên tập Y_i' không bao giờ rỗng.

Bước 3 : Chọn đỉnh đích y

Trong tập Y_i', ta chọn một đỉnh y sao cho giá trị độ lệch hướng $d_r(y)$ là nhỏ nhất. Tương tự như cách chọn đỉnh nguồn x, việc ưu tiên chọn đỉnh y có độ lệch nhỏ giúp giảm khả năng chọn các đỉnh "xa gốc", vốn dễ gây xung đột hoặc chồng lấn với các đường đi khác. Điều này góp phần duy trì cấu trúc ổn định cho toàn bộ quá trình xây dựng phép ghép các đường đi có hướng, đồng thời hạn chế sự xuất hiện của các đường đi lệch hướng yếu (weakly biased path pairs).

Bước 4 : Tạo đường đi P_i và cập nhật lại tập đỉnh

Vì đồ thị cơ sở là một cây (polytree), nên giữa x và y chỉ tồn tại một đường đi duy nhất trong đồ thị cơ sở T^{und}. Hơn nữa, do các cung đã được định hướng, đường

đi từ $x \to y$ cũng là duy nhất nếu tồn tại. Ta xác định đường đi P_i từ x đến y, sử dụng chính các cung đã kiểm tra ở bước 2. Sau đó, ta gán đường đi đó là P_i và thêm vào tập các đường đã tạo:

$$\mathcal{P} := \mathcal{P} \cup \{P_i\}$$

Trước khi tiến hành bước tiếp theo, ta cần cập nhật lại tập đỉnh, cụ thể ta sẽ loại bỏ x và y khỏi hai tập nguồn và đích.

$$X_{i+1} := X_i \setminus \{x\}, \quad Y_{i+1} := Y_i \setminus \{y\}$$

Bước 5 : Duy trì bất biến và lặp lại

Vì ta chỉ sử dụng các cung e thoả mãn điều kiện $w(e; X_i, Y_i) > 0$ khi tạo đường đi P_i, nên việc loại bỏ hai đỉnh x và y khỏi các tập X_i và Y_i tương ứng sẽ chỉ làm giảm trọng số của mỗi cung thuộc P_i đi đúng một đơn vị.

$$\Rightarrow w(e; X_{i+1}, Y_{i+1}) \geq 0 \quad \forall e,$$

Điều này đảm bảo rằng bất biến của thuật toán tiếp tục được duy trì cho bước tiếp theo. Thuật toán sẽ tiếp tục lặp lại các bước cho đến khi :

$$X_i = \emptyset, \quad Y_i = \emptyset$$

Khi đó tất cả các đỉnh trong $f(I^s)$ và $g(I^t)$ đều đã được ghép cặp một cách hoàn chỉnh.

5.2.4 Chứng minh rằng phép ghép các đường đi có hướng thu được không chứa cặp đường đi lệch hướng yếu và tính độ phức tạp thuật toán

Sau khi thuật toán kết thúc và xây dựng được một phép ghép các đường đi có hướng $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$, ta cần chứng minh rằng tập này không chứa bất kỳ cặp không chứa bất kỳ cặp đường đi lệch hướng yếu (weakly biased path pairs). Giả sử tồn tại hai đường đi P_i và P_j với $i < j$ có một đỉnh chung v (định nghĩa weakly biased path pair yêu cầu chỉ có đỉnh chung, không nhất thiết là đỉnh nội bộ). Theo thuật toán, đường đi P_i được xây dựng trước P_j, do đó đỉnh $s(P_i) \in X_i$ đã được chọn trước đỉnh $s(P_j)$. Tại mỗi bước i, thuật toán luôn chọn $x = s(P_i) \in X_i$ sao cho:

$$d_r(x) = d_r(s(P_i)) \leq d_r(s(P_j)) \qquad \forall j > i$$

Ta có bất đẳng thức quan trọng: $d_r(s(P_i)) \leq d_r(s(P_j))$

Cho gốc $r \in V$, với mọi đỉnh $v \in V$, ta định nghĩa:

$$d_r(v) = \mathrm{fwd}(\overrightarrow{P_{r,v}}) - \mathrm{rev}(\overrightarrow{P_{r,v}})$$

Với các đỉnh $u, v \in V$ cùng nằm trên đường đi từ r đến v, ta có:

$$\mathrm{dist}(u, v) = d_r(v) - d_r(u)$$

Áp dụng cho $s(P_i), s(P_j), v$, ta suy ra:

$$\mathrm{dist}(s(P_i), v) = -d_r(s(P_i)) + d_r(v)$$

$$\mathrm{dist}(s(P_j), v) = -d_r(s(P_j)) + d_r(v)$$

Do đó, nếu $d_r(s(P_i)) \leq d_r(s(P_j))$ thì:

$$\mathrm{dist}(s(P_i), v) \geq \mathrm{dist}(s(P_j), v)$$

Vì v là đỉnh chung của P_i và P_j, nên $t(P_j) \in Y_i'$. Theo cách chọn y tại bước i, ta có:

$$d_r(t(P_i)) \leq d_r(t(P_j))$$

Do đó, tương tự như phần nguồn, ta có:

$$\mathrm{dist}(v, t(P_i)) = -d_r(v) + d_r(t(P_i)) \leq -d_r(v) + d_r(t(P_j)) = \mathrm{dist}(v, t(P_j))$$

Từ hai bất đẳng thức đã chứng minh:

$$\mathrm{dist}(s(P_i), v) \geq \mathrm{dist}(s(P_j), v) \quad \text{và} \quad \mathrm{dist}(v, t(P_i)) \leq \mathrm{dist}(v, t(P_j))$$

ta có thể nhận thấy rằng đường từ $s(P_j)$ đến v ngắn hơn đường từ $s(P_i)$ đến v, nhưng đoạn từ v đến $t(P_j)$ lại dài hơn hoặc bằng đoạn từ v đến $t(P_i)$. Điều này mâu thuẫn với định nghĩa của một *weakly biased path pair*, vốn yêu cầu đường đi P_i phải "lệch hơn" và bao trùm so với P_j tại điểm v. Từ đó ta có thể kết luận rằng P_i và P_j không tạo thành một weakly biased path pair.

Để tính tổng độ phức tạp thời gian của thuật toán xây dựng chuỗi tái cấu hình, ta xét ba bước chính. Thứ nhất, với mỗi cặp đỉnh $(x, y) \in f(I^s) \times g(I^t)$, ta cần xây dựng một đường đi duy nhất P_i trong cây, do cấu trúc không chu trình của đồ thị polytree đảm bảo tính đơn nhất của đường đi. Mỗi đường đi này có độ dài tối đa là $O(|V|)$, nên việc xây một đường đi mất thời gian $O(|V|)$. Thứ hai, vì số lượng token là $k = |I^s| = |I^t| \leq |V|$, nên tổng chi phí cho việc xây tất cả các đường đi là:

$$T_1 = k \times O(|V|) = O(|V| \cdot k) \leq O(|V|^2)$$

Thứ ba, như đã trình bày, thời gian tính toán hai ánh xạ f và g là:

$$T_2 = O(|V|)$$

Do đó, tổng thời gian của thuật toán là:

$$T_{\text{tổng}} = T_1 + T_2 = O(|V|^2) + O(|V|) = \boxed{O(|V|^2)}$$

Đây là một kết quả tối ưu cho trường hợp đồ thị có cấu trúc cây định hướng.

5.3 Các bước cơ sở xây dựng thuật toán tuyến tính và minh họa trực quan

5.3.1 Thuật toán quyết định - Decision algorithm

Trước khi xây dựng cụ thể một chuỗi tái cấu hình từ I^s đến I^t một bước quan trọng và cần thiết là kiểm tra tính khả thi của việc tái cấu hình – tức là liệu có tồn tại một chuỗi di chuyển nào thoả mãn điều kiện không vi phạm tính độc lập tại mọi bước hay không. Để trả lời câu hỏi này, nhóm tác giả [1] đã đề xuất một thuật toán quyết định tuyến tính, vận hành hiệu quả trên đồ thị có cấu trúc polytree – nơi đồ thị vô hướng cơ sở là một cây.

Thuật toán hoạt động dựa trên việc:

- Phân tích "dòng chảy" chênh lệch giữa các vị trí token ban đầu và vị trí đích.
- Loại bỏ sớm các token không thể di chuyển.
- Kiểm tra điều kiện cần đủ dựa trên trọng số định nghĩa trên các cung.

Dưới đây là mô tả chi tiết năm bước chính của thuật toán, mỗi bước gắn liền với một mục tiêu cụ thể nhằm giảm không gian bài toán và xác định khả năng tái cấu hình một cách chính xác và hiệu quả.

Bước 1 - **Tiền xử lý**. Việc đầu tiên đó chính là kiểm tra điều kiện tiên quyết $|I^s| = |I^t|$, tức là số lượng token trong tập khởi đầu và tập đích phải bằng nhau. Sau đó xác minh tính hợp lệ của hai tập I^s và I^t: đảm bảo rằng cả hai đều là tập độc lập trong đồ thị T – nghĩa là không tồn tại hai đỉnh bất kỳ trong I^s (hoặc I^t) có liên kết trực tiếp thông qua một cung.

Đây là bước kiểm tra sơ bộ để loại bỏ nhanh các trường hợp vô nghiệm hiển nhiên. Nếu hai tập có số lượng token khác nhau, hoặc một trong hai tập chứa các đỉnh kề nhau thì không thể tồn tại bất kỳ chuỗi tái cấu hình hợp lệ nào, do ngay từ đầu cấu

hình đã không thỏa mãn điều kiện độc lập – vốn là ràng buộc bắt buộc trong bài toán Directed Token Sliding. Do đó, nếu kiểm tra không đạt, thuật toán có thể kết luận NO ngay mà không cần thực hiện các bước tiếp theo.

Bước 2: **Tính trọng số** $w(e)$. Các thao tác thực hiện của bước này như sau:

- Thực hiện một lượt duyệt cây (DFS hoặc BFS đều được) trên đồ thị T, nhằm xác định tập con C_e^- tức là tập đỉnh nằm trong thành phần bị tách ra nếu loại bỏ cung e.
- Với mỗi cung $e \in A$, ta tính trọng số lệch $w(e; I^s, I^t)$ theo công thức:

$$w(e) = \left|C_e^- \cap I^s\right| - \left|C_e^- \cap I^t\right|$$

  C_e^- là tập các đỉnh thuộc về thành phần liên thông (connected component) chứa đỉnh *đuôi* (tail) của cung e, sau khi loại bỏ cung e khỏi đồ thị.

Trọng số $w(e)$ phản ánh sự lệch dòng di chuyển của token qua cung e tức là cung đó sẽ "chịu trách nhiệm" chuyển bao nhiêu token từ phía I^s sang I^t. Theo **Bổ đề 4.2**, điều kiện $w(e) \geq 0 \quad \forall e \in A$ là điều kiện cần và đủ để tồn tại một phép ghép các đường đi có hướng (Directed Path Matching – DPM), đảm bảo rằng các token có thể dịch chuyển mà không gây "quá tải" trên bất kỳ cung nào trong đồ thị.

Bước 3 - **Phát hiện phần cố định**. Bước này nhằm xác định các token hoặc cung không tham gia vào quá trình tái cấu hình, từ đó loại bỏ trước để giảm quy mô bài toán và phát hiện sớm các trường hợp vô nghiệm. Áp dụng **Định lý 4.5** một token được gọi là cố định nếu (rigid tokens):

$$v \in R \;\Leftrightarrow\; v \in I^s \cap I^t \quad \text{và} \quad w(e) = 0 \quad \forall e \in \Gamma(v)$$

Tức là token nằm ở vị trí ban đầu và vị trí cuối giống nhau, và không có dòng lệch nào qua các cung kề với nó. Sau đó ta cần xác định tập cung giới hạn di chuyển (blocking arcs). Áp dụng **Định lý 4.7** một cung $e = (u, v)$ được gọi là blocking arcs nếu thỏa mãn:

$$w(e) = 1 \quad \text{và} \quad w(e') = 0 \quad \forall e' \in \Gamma(u) \cup \Gamma(v) \setminus \{e\}$$

Nghĩa là cung đó chịu trách nhiệm duy nhất cho việc chuyển một token qua lại giữa hai đỉnh u, v, và không có dòng nào khác liên quan đến hai đỉnh đó.

Việc nhận diện sớm các phần tử không tác động giúp rút gọn bài toán, tránh phải xử lý những phần "cứng nhắc" không liên quan đến quá trình chuyển đổi. Nếu tồn tại một *rigid token* $v \in R$ mà lại kề với một đỉnh u sao cho dòng lệch $w(e) > 0$ với

$e = (v, u)$ hoặc $e = (u, v)$, thì theo **Hệ quả 4.6**, chuỗi tái cấu hình là bất khả thi. Lý do là không có cách nào để token tại v di chuyển qua hoặc tránh xung đột với token khác, do v vừa cố định vừa bị "áp lực" bởi dòng lệch. Khi đó, thuật toán có thể kết luận *NO* ngay lập tức mà không cần xét tiếp các bước còn lại.

Bước 4 - Rút gọn polyforest T'. Ở bước này, ta xây dựng một phiên bản rút gọn của đồ thị ban đầu, gọi là $T' = (V, A')$, bằng cách loại bỏ các cung không cần thiết để tập trung vào phần có khả năng tái cấu hình. Ta tiến hành loại bỏ các cung theo hai quy tắc:

- Loại bỏ các cung kề với các đỉnh cố định (rigid tokens R)
  Nếu một đỉnh $v \in R$ (tức là một *rigid token*), thì tất cả các cung kề với v — nghĩa là mọi cung có v là đỉnh đầu hoặc đỉnh đuôi — sẽ bị loại khỏi tập cung A'. Những token cố định không di chuyển trong suốt quá trình, nên các cung kề với chúng không có giá trị trong việc kiểm tra khả năng tái cấu hình.
- Loại bỏ các cung không thuộc tập blocking B nhưng lại kề một đầu mút của cung trong B
  Nếu một cung $e = (u, v) \notin B$ nhưng u hoặc v thuộc vào một *blocking arc* khác, thì e cũng sẽ bị loại khỏi tập cung A'. Các blocking arc là những "đường đi duy nhất" cho một token. Mọi cung kề với chúng nhưng không phải blocking đều không được phép can thiệp (nếu có sẽ làm hỏng tính duy nhất), nên cần loại bỏ để tránh nhiễu.

Việc loại bỏ các cung không liên quan giúp tạo ra một mô hình gọn nhẹ, chỉ giữ lại phần có thể ảnh hưởng đến việc tái cấu hình. Điều này làm cho việc kiểm tra điều kiện khả thi sau đó nhanh hơn và chính xác hơn. Sau khi rút gọn, mỗi thành phần còn lại trong T' là độc lập hoàn toàn, không có kết nối với các phần bị loại. Điều này giúp ta có thể kiểm tra khả năng tái cấu hình trên từng thành phần riêng biệt, rồi kết luận cho toàn bộ bài toán. Theo **Định lý 4.8**, việc rút gọn như trên không làm thay đổi kết luận YES/NO của bài toán – tức là nếu tồn tại chuỗi tái cấu hình trong T, thì cũng tồn tại trong T' và ngược lại.

Bước 5 -Kiểm tra điều kiện cuối cùng. Sau khi đã xây dựng đồ thị rút gọn $T' = (V, A')$, ta tiến hành duyệt qua toàn bộ các cung $e \in A'$ và kiểm tra trọng số lệch $w(e)$ trên từng cung. Nếu tồn tại ít nhất một cung $e \in A'$ sao cho $w(e) < 0$, thì thuật toán sẽ kết luận *no-instance*, tức là không tồn tại chuỗi tái cấu hình hợp lệ từ I^s đến I^t. Điều kiện trọng số không âm $w(e) \geq 0$ trên mọi cung e là điều kiện cần và đủ để

tồn tại một phép ghép các đường đi có hướng (Directed Path Matching – DPM), theo **Bổ đề 4.2**. Khi ta đã loại bỏ phần cố định R và các *blocking arcs* B ở các bước trước, thì điều kiện tồn tại một phép ghép các đường đi có hướng (DPM) là tương đương với việc tồn tại một chuỗi *Token Sliding* hợp lệ giữa hai tập I^s và I^t. Do đó, việc kiểm tra tất cả các cung trong T' có thỏa mãn điều kiện $w(e) \geq 0$ hay không trở thành bước quyết định cuối cùng: đây chính là điểm chốt để thuật toán trả lời *yes-instance* hoặc *no-instance* cho toàn bộ bài toán tái cấu hình.

5.3.2 Thuật toán dựng chuỗi - Construction algorithm

Sau khi kết thúc bước B4 của thuật toán quyết định, giả sử ta đã xây dựng được một *polyforest* rút gọn $T' = (V, A')$ thỏa mãn các điều kiện sau:

- Không còn *rigid token* (token cố định).
- Không còn *blocking arc* (token chỉ có đúng một khả năng di chuyển).
- Trọng số $w(e; I^s, I^t) \geq 0$ với mọi cung $e \in A'$.

Khi đó, theo **Bổ đề 4.2**, ta đã bảo đảm sự tồn tại của một *phép ghép các đường đi có hướng* (Directed Path Matching – DPM) từ I^s đến I^t. Mục tiêu tiếp theo của chúng ta sẽ là tạo tường minh một chuỗi tái cấu hình (DTS sequence) từ I^s đến I^t, sao cho mọi bước trung gian đều là một tập độc lập hợp lệ trong đồ thị ban đầu.

Bước 1 - **Ghép DPM thông qua BFS một chiều**. Bước này nhằm bảo đảm rằng ta có thể ghép mỗi token từ tập khởi đầu I^s với một vị trí đích trong I^t, theo các đường đi có hướng hợp lệ, không gây xung đột về dòng chảy. Việc này được thực hiện bằng thuật toán BFS một chiều.

- Khởi chạy BFS một chiều:
  - Bắt đầu từ tất cả các đỉnh thuộc I^s (nguồn).
  - Duyệt theo các cung có trọng số dương $w(e) > 0$ trong đồ thị T'.
  - Đảm bảo rằng chỉ các đường đi "có thể sử dụng được" (tức là không vượt quá dung lượng dòng chảy) mới được xét đến.
- Gán ánh xạ khi gặp đỉnh đích:
  - Mỗi khi gặp một đỉnh $y \in I^t$ lần đầu tiên, ta ghi nhận đỉnh $x \in I^s$ đã sinh ra nó qua BFS.
  - Từ đó xác lập ánh xạ $\pi(x) = y$, và lưu lại đường đi P_x^* từ x đến y.
- Lặp lại cho đến khi ánh xạ hoàn tất:

– Quá trình lặp tiếp tục cho đến khi mỗi đỉnh trong I^s được ghép với đúng một đỉnh trong I^t, tạo nên một ánh xạ song ánh $\pi : I^s \to I^t$.

Ta thu được tập các đường đi có hướng $\mathcal{P}^* = \{P_1^*, P_2^*, \ldots, P_k^*\}$, trong đó mỗi đường đi nối một đỉnh từ I^s đến một đỉnh tương ứng trong I^t. Mỗi đường đi chỉ sử dụng các cung có $w(e) > 0$, do đó không gây xung đột và bảo đảm điều kiện phép ghép các đường đi (DPM) theo **Bổ đề 4.2**.

Bước 2 – Xây dựng đồ thị phụ thuộc token G. Bước này nhằm xây dựng một đồ thị phụ thuộc (token dependency graph) phản ánh mối quan hệ giữa các token trong quá trình tái cấu hình. Mỗi cung trong đồ thị thể hiện rằng một token phải di chuyển trước, để đảm bảo token khác có thể di chuyển an toàn mà không vi phạm tính độc lập. Đồ thị này sau đó sẽ được dùng để sắp xếp topo và lên kế hoạch di chuyển cụ thể.

- Định nghĩa đồ thị phụ thuộc token G
  – Với mỗi token tương ứng với một đường đi $P_i^* \in P^*$ từ $I^s \to I^t$, ta tạo một đỉnh i trong đồ thị phụ thuộc G.
  – Gọi $s(P_i^*)$ và $t(P_i^*)$ lần lượt là đỉnh đầu và đỉnh cuối của đường đi P_i^*.
- Tạo cung phụ thuộc (i, j)
  Với mỗi cặp đường đi P_i^* và P_j^*, nếu hai đường đi giao nhau tại một đỉnh v và

$$\text{dist}(s(P_i^*), v) < \text{dist}(s(P_j^*), v)$$

  thì thêm cung (i, j) vào đồ thị phụ thuộc G. Ý nghĩa đó là token i cần phải đi trước token j để tránh xung đột tại đỉnh v.
  Kết quả thu được đồ thị phụ thuộc $G = (V_G, E_G)$, trong đó mỗi đỉnh tương ứng với một token, và mỗi cung biểu diễn thứ tự cần thiết giữa các thao tác trượt token. Đồ thị này sẽ là cơ sở để thực hiện sắp xếp topo trong bước kế tiếp nhằm xây dựng chuỗi tái cấu hình DTS hợp lệ.

Bước 3 – Sắp xếp topo và xác định thứ tự di chuyển. Bước này nhằm xác định trình tự di chuyển hợp lệ cho các token thông qua thuật toán sắp xếp topo trên đồ thị phụ thuộc token đã xây dựng ở bước trước. Thứ tự này đảm bảo rằng không có sự xung đột giữa các token trong quá trình tái cấu hình, tức là không có token nào bị chặn bởi một token khác chưa di chuyển.

- Thực hiện sắp xếp topo
  – Áp dụng thuật toán sắp xếp topo chuẩn (ví dụ: thuật toán Kahn hoặc DFS có gán nhãn kết thúc) trên đồ thị phụ thuộc G.

- Kết quả là một thứ tự $i_1, i_2, \ldots, i_k$ ứng với các đường đi $P^*_{i_1}, P^*_{i_2}, \ldots, P^*_{i_k}$ từ $I^s \to I^t$.

- Ý nghĩa của thứ tự topo

  Nếu tồn tại cung $(i, j) \in E_G$, thì token i phải di chuyển trước token j. Thứ tự topo này cho ta một kế hoạch an toàn: token nào được "giải phóng" trước thì sẽ di chuyển trước, đảm bảo không có giao nhau tại các đỉnh trung gian.

- Ứng dụng trong việc xây dựng chuỗi DTS
  - Duyệt qua các token theo thứ tự đã sắp xếp.
  - Với mỗi token i, thực hiện thao tác trượt token từ $s(P^*_i)$ đến $t(P^*_i)$ theo đúng đường đi P^*_i.
  - Do các đường đi đã được đảm bảo không xung đột (nhờ bước ghép DPM và xây dựng đồ thị phụ thuộc), nên chuỗi thao tác giữ được tính chất độc lập sau mỗi bước.

Kết quả cuối cùng sẽ là chuỗi các thao tác di chuyển token thu được chính là một chuỗi tái cấu hình hợp lệ, nối từ I^s đến I^t, trong đó mỗi bước là một thao tác trượt token duy nhất, không vi phạm điều kiện độc lập tại bất kỳ thời điểm nào.

Bước 4 – Di chuyển token tuần tự theo thứ tự topo. Bước này nhằm thực thi chuỗi tái cấu hình, tức là di chuyển từng token từ đỉnh xuất phát đến đỉnh đích theo các đường đi đã xác định ở bước 1, theo đúng thứ tự topo đã tính được ở bước 3. Điều này đảm bảo rằng tại mỗi thời điểm, chỉ một token được di chuyển, và tính chất độc lập của tập token luôn được duy trì.

- Lặp theo thứ tự topo
  - Gọi $\pi(1), \pi(2), \ldots, \pi(k)$ là thứ tự topo của các token, tương ứng với các đường đi $P^*_{\pi(1)}, P^*_{\pi(2)}, \ldots, P^*_{\pi(k)}$.
  - Với mỗi $i = 1$ đến k, thực hiện thao tác trượt token từ đỉnh $s(P^*_{\pi(i)})$ đến $t(P^*_{\pi(i)})$ theo đúng đường đi $P^*_{\pi(i)}$.
- Từng bước trượt
  - Mỗi đường đi $P^*_{\pi(i)}$ là có hướng, không chứa xung đột, và tại thời điểm thực hiện thao tác, các token khác chưa chiếm dụng bất kỳ đỉnh nào trên đường đi này.
  - Vì các token trước đó trong thứ tự topo đã hoàn tất việc di chuyển, nên token thứ i không gây ra giao cắt hoặc xung đột với các token còn lại.

- Duy trì tính độc lập
  - Do mỗi bước chỉ trượt một token, và các thao tác diễn ra theo đúng thứ tự phụ thuộc đã phân tích, nên mọi cấu hình trung gian đều là tập độc lập hợp lệ.
  - Không có tình huống nào trong đó hai token cùng xuất hiện tại hai đỉnh kề nhau trong cùng một thời điểm.

Kết quả là sau k bước, toàn bộ token được trượt thành công từ I^s sang I^t thông qua các đường đi hợp lệ. Chuỗi thao tác thu được là một chuỗi tái cấu hình từ I^s đến I^t, trong đó mỗi bước chỉ có một token di chuyển, và luôn duy trì tính chất của một tập độc lập.

Bước 5 – Ghép các thành phần của T' để tạo chuỗi DTS hoàn chỉnh. Trong trường hợp đồ thị $T' = (V, A')$ sau bước rút gọn có nhiều thành phần liên thông yếu (do loại bỏ các *rigid tokens* và các cung bị chặn), bước này thực hiện việc ghép các chuỗi tái cấu hình DTS riêng biệt trên từng thành phần để hình thành một chuỗi tái cấu hình chung duy nhất từ tập ban đầu I^s đến tập đích I^t.

- Xử lý từng thành phần liên thông yếu
  - Đồ thị rút gọn T' có thể phân rã thành các thành phần nhỏ hơn $T'_1, T'_2, \ldots, T'_r$, mỗi thành phần là một *polytree* con độc lập.
  - Với mỗi T'_i, ta xác định:

    $$I^s_{(i)} = I^s \cap V(T'_i), \qquad I^t_{(i)} = I^t \cap V(T'_i)$$

    là tập token ban đầu và đích tương ứng trong thành phần T'_i.
  - Áp dụng lại đầy đủ các bước từ 1 đến 4 đã mô tả trước đó để xây dựng chuỗi tái cấu hình DTS riêng từ $I^s_{(i)}$ đến $I^t_{(i)}$ trong thành phần T'_i.
- Ghép các chuỗi riêng thành chuỗi chung
  - Do các thành phần không liên thông (sau bước B3 đã loại bỏ các cung có thể gây xung đột), các token thuộc các thành phần khác nhau không ảnh hưởng lẫn nhau.
  - Vì vậy, có thể ghép nối tuần tự các chuỗi DTS con thu được trong từng thành phần để tạo thành một chuỗi tái cấu hình tổng thể từ I^s đến I^t trên toàn bộ đồ thị.
- Đảm bảo tính đúng đắn
  - Mỗi bước trong chuỗi tái cấu hình tổng thể chỉ di chuyển một token duy nhất.
  - Tính chất tập độc lập luôn được duy trì vì các thành phần đã được xử lý tách biệt, không chia sẻ cung chung hay đỉnh kề nhau.

Việc xử lý từng thành phần liên thông yếu và ghép nối chuỗi tái cấu hình của chúng giúp thuật toán mở rộng được cho các trường hợp phức tạp, trong đó T' không còn là một cây liên thông duy nhất. Đây chính là bước cuối cùng để xây dựng thành công một chuỗi tái cấu hình hợp lệ (DTS) từ I^s đến I^t.

5.3.3 Yes-instance trên đồ thị có hai nhánh hội tụ về đỉnh chung

Trong mục này ta xây dựng một ví dụ mà ở đó ta sẽ theo dõi từng bước của hai thuật toán (Decision → Construction) trên một polytree rất nhỏ để thấy rõ lý do và kết quả của mỗi thao tác.

A.Đặt vấn đề

Đồ thị Polytree T có năm đỉnh và bốn cung, tả hai nhánh cùng đổ vào đỉnh c :

$$a \to b, \quad b \to c, \quad e \to d, \quad d \to c.$$

Tập ban đầu: $I^s = \{a, e\}$, tập đích: $I^t = \{b, d\}$. Hai tập I^s và I^t có cùng kích thước và rõ ràng là không có đỉnh nào kề nhau, do đó đều là các tập độc lập hợp lệ.

B. Thuật toán quyết định

B1 - Tiền xử lý

– Kiểm tra điều kiện kích thước

$$|I^s| = |I^t| = 2 \Rightarrow \text{hợp lệ.}$$

– Xác nhận tính độc lập:

∗ $I^s = \{a, e\}$: a không kề e trong đồ thị có hướng.

∗ $I^t = \{b, d\}$: b không kề d.

$\Rightarrow$ Cả hai tập I^s và I^t đều là tập độc lập hợp lệ. Mục đích của bước này là xác định xem có thể loại bỏ *instance* sớm do vi phạm điều kiện cơ bản (khác kích thước, chứa cạnh trong tập). Nếu không, chuyển sang bước tiếp theo. Như vậy điều kiện ban đầu thỏa mãn $\Rightarrow$ tiếp tục bước B1.

B2 - Tính trọng số $w(e)$

Trọng số được định nghĩa bởi:

$$w(e; I^s, I^t) = |C_e^- \cap I^s| - |C_e^- \cap I^t| \tag{5.1}$$

trong đó:

– C_e^- là thành phần liên thông yếu chứa đỉnh đuôi của cung e sau khi loại bỏ e khỏi đồ thị T.

– I^s, I^t lần lượt là tập token ban đầu và tập token đích.

Đồ thị T là một có hai nhánh hội tụ về đỉnh chung:

$$a \xrightarrow{e_1} b \xrightarrow{e_2} c \xleftarrow{e_3} d \xleftarrow{e_4} e$$

Tập ban đầu: $I^s = \{a, e\}$, Tập đích: $I^t = \{b, d\}$.

Cung e	**Thành phần** C_e^-	$\lvert C_e^- \cap I^s \rvert$	$\lvert C_e^- \cap I^t \rvert$	$w(e)$
$e_1 : a \to b$	$\{a\}$	1 (a)	0	1
$e_2 : b \to c$	$\{a, b\}$	1 (a)	1 (b)	0
$e_3 : d \to c$	$\{e, d\}$	1 (e)	1 (d)	0
$e_4 : e \to d$	$\{e\}$	1 (e)	0	1

Bảng 5.2: Bảng tính trọng số dòng chảy $w(e)$ cho đồ thị T

Để xác định tập C_e^- cho một cung $e = (u, v)$, ta thực hiện việc loại bỏ cung đó khỏi đồ thị và xét thành phần liên thông chứa đỉnh đuôi u. Cụ thể:

– Khi bỏ cung $a \to b$, đỉnh a bị tách rời, nên $C_{e_1}^- = \{a\}$.

– Bỏ cung $b \to c$, đuôi b vẫn thuộc nhánh trái cùng với a, nên $C_{e_2}^- = \{a, b\}$.

– Bỏ cung $d \to c$, đuôi d vẫn thuộc nhánh phải cùng với e, nên $C_{e_3}^- = \{e, d\}$.

– Bỏ cung $e \to d$, đỉnh e tách riêng, nên $C_{e_4}^- = \{e\}$.

Khi tính trọng số theo định nghĩa:

$$w(e) = |C_e^- \cap X| - |C_e^- \cap Y|,$$

ta thu được các giá trị:

$$w(a \to b) = 1, \quad w(b \to c) = 0, \quad w(d \to c) = 0, \quad w(e \to d) = 1.$$

Vì tất cả trọng số đều không âm, đồ thị T thoả mãn điều kiện dòng chảy theo **Bổ đề 4.2**.Tổng các giá trị $w(e)$ là:

$$\sum_{e \in A} w(e) = 1 + 0 + 0 + 1 = 2,$$

chính là độ dài bất biến của mọi chuỗi tái cấu hình giữa hai trạng thái hợp lệ, theo **Hệ quả 4.3**.

B3 - Phát hiện phần cố định

Để xác định các *rigid token* và *blocking arc* theo đúng định nghĩa trong bài báo (**Bổ đề 4.5 và 4.7**), ta tiến hành phân tích như sau.

Một đỉnh được xem là rigid token nếu tất cả các cung kề với nó đều có trọng số $w = 0$. Xét các đỉnh đang mang token là a, e, b và d, ta thấy mỗi đỉnh đều có ít nhất một cung kề có trọng số $w = 1$, không đỉnh nào thỏa điều kiện rigid token.

Kết luận: Tập rigid token $R = \varnothing$.

Đối với blocking arc, một cung $e = (u, v)$ được xem là blocking nếu nó có trọng số $w(e) = 1$ và tất cả các cung khác kề với u hoặc v đều có trọng số bằng 0.

- Xét cung $a \to b$: ta có $w = 1$, không có cung nào khác kề a, và cung kề b là $b \to c$ có $w = 0 \Rightarrow$ thoả điều kiện.
- Xét cung $e \to d$: $w = 1$, không có cung khác kề e, và cung $d \to c$ có $w = 0 \Rightarrow$ cũng thoả mãn.
- Hai cung còn lại đều có trọng số $w = 0$ nên không được xét.

Kết luận: Tập các blocking arc là $B = \{a \to b,\ e \to d\}$.

Như vậy, trong trường hợp này ta có $B \neq \varnothing$ và $R = \varnothing$.

B4 - Rút gọn polyforest

Theo quy tắc rút gọn, ta thực hiện hai bước để đơn giản hoá đồ thị trước khi áp dụng thuật toán tái cấu hình.

- **Bước 1:** Loại bỏ tất cả các cung kề với các đỉnh thuộc tập rigid token R. Tuy nhiên, trong trường hợp này $R = \varnothing$ nên bước này được bỏ qua.
- **Bước 2:** Với mỗi blocking arc $e \in B$, ta giữ lại chính cung e nhưng xoá toàn bộ các cung khác kề với hai đầu mút của e nếu chúng không thuộc B.

Cụ thể, với hai blocking arc là $a \to b$ và $e \to d$, ta loại bỏ các cung có trọng số bằng 0 kề với các đầu mút:

- Cung $b \to c$ (kề với b) bị loại bỏ.
- Cung $d \to c$ (kề với d) cũng bị loại bỏ.

Sau khi áp dụng quy tắc rút gọn này, đồ thị bị chia thành ba thành phần rời nhau:

- Thành phần 1: $\{a, b\}$ chứa blocking arc $a \to b$.
- Thành phần 2: $\{e, d\}$ chứa blocking arc $e \to d$.
- Thành phần 3: đỉnh đơn lẻ c, vì mọi cung kề với nó đã bị loại bỏ.

Do không còn rigid token và cũng không còn các cung ngược chiều, mỗi thành phần giờ đã hoàn toàn độc lập. Thuật toán có thể tiếp tục xử lý riêng biệt từng phần này.

B5 - Kiểm tra cuối

Trên đồ thị rút gọn $T' = B \cup \{\text{cô lập } c\}$, chỉ còn lại hai cung thuộc tập blocking arc là $a \to b$ và $e \to d$, cùng với đỉnh đơn lẻ c. Tất cả các cung còn lại trong T' đều có trọng số $w(e) \geq 0$, do đó thỏa mãn điều kiện của **Bổ đề 4.2**.

Theo đó, điều này tương đương với việc tồn tại một *Directed Path Matching* (DPM) từ tập khởi đầu I^s đến tập đích I^t, tức là tồn tại một ánh xạ các token từ trạng thái ban đầu sang trạng thái đích thông qua các đường đi có hướng, không gây xung đột.

Như vậy, bộ giải Decision có thể kết luận đây là một *yes-instance* và chuyển tiếp sang thuật toán Construction để tiến hành dựng cụ thể chuỗi tái cấu hình hợp lệ.

C. Thuật toán xây dựng chuỗi tái cấu hình

C1 - Ghép DPM bằng BFS thuận chiều

Tại bước này, thuật toán tiến hành thiết lập ánh xạ giữa các đỉnh trong tập khởi đầu I^s và tập đích I^t bằng cách thực hiện BFS thuận chiều từ mỗi đỉnh nguồn. Cụ thể, từ đỉnh $a \in I^s$, ta thực hiện BFS và đỉnh đầu tiên thuộc I^t gặp phải là b, thu được đường đi duy nhất $P_a^* : a \to b$. Tương tự, từ đỉnh e, ta thu được $P_e^* : e \to d$.

Vì đồ thị T là một *polytree có hướng*, nên giữa hai đỉnh bất kỳ tồn tại nhiều nhất một đường đi có hướng. Do đó, phép BFS "gặp đích đầu tiên"luôn tạo ra đúng một cặp duy nhất giữa một đỉnh nguồn và một đích, đảm bảo ánh xạ là đơn ánh.

Nguồn (I^s)	**BFS thuận chiều**	**Đích đầu tiên gặp ($\subset I^t$)**	**Đường đi duy nhất thu được**
a	$a \to b$	b	$P_a^* : a \to b$
e	$e \to d$	d	$P_e^* : e \to d$

Bảng 5.3: Kết quả BFS thuận chiều từ mỗi đỉnh trong I^s

Đồng thời, hai đường đi P_a^* và P_e^* không chia sẻ cạnh hay đỉnh nội bộ, nên việc thực hiện di chuyển các token theo các đường đi này là độc lập và không gây xung đột. Tập các đường đi thu được là:

$$P^* = \{P_a^* = a \to b, \quad P_e^* = e \to d\}.$$

Đây là mục tiêu của bước C1: xác lập một ánh xạ song ánh giữa I^s và I^t, sao cho mỗi token nguồn được gán duy nhất một đích. Điều này cung cấp *"xương sống"* cho các bước tiếp theo như: xây dựng đồ thị phụ thuộc token, sắp xếp topo, và lập kế hoạch di chuyển — loại bỏ hoàn toàn sự mơ hồ trong quá trình tái cấu hình.

C2 - Xây dựng đồ thị phụ thuộc token G

Định nghĩa quan hệ "token i cản token j" (trích §4.3.3) nêu rằng cho hai đường đi P_i^*, P_j^*. Ký hiệu s_i, t_i là đỉnh đầu/cuối của P_i^*, tương tự với s_j, t_j. Ta nói $i \dashleftarrow j$ nếu một trong hai điều kiện sau xảy ra:

- (1) s_i kề một đỉnh của P_j^* (token i "nhìn sát" đường đi của j),
- (2) t_j kề một đỉnh của P_i^* (đích của j chắn lối token i).

Nếu xảy ra, ta thêm cung (i, j) vào đồ thị phụ thuộc G.

Cặp token	**Kiểm tra S-block**	**Kiểm tra T-block**	**Kết luận**
a vs e	$s_a = a$ chỉ kề b; $P_e^* = \{e, d\}$ $\rightarrow$ không kề	$t_e = d$ kề c, e; $P_a^* = \{a, b\} \rightarrow$ không kề	Không thêm cung
e vs a	$s_e = e$ chỉ kề d; $P_a^* = \{a, b\}$ $\rightarrow$ không kề	$t_a = b$ kề a, c; $P_e^* = \{e, d\} \rightarrow$ không kề	Không thêm cung

Bảng 5.4: Xác định quan hệ cản giữa các token

Áp dụng vào ví dụ với hai token xuất phát từ a và e, ta thấy không tồn tại mối liên kết kề giữa đỉnh xuất phát của một token và đường đi của token còn lại, cũng như giữa đỉnh đích của một token và đường đi của token kia. Cụ thể:

- a chỉ kề b, trong khi $P_e^* = e \rightarrow d$ không chứa đỉnh nào kề với a.
- d và b cũng không kề nhau theo đồ thị vô hướng nền.

Do đó, không có cung nào được thêm vào G, và đồ thị phụ thuộc thu được là:

$$G = (\{a, e\}, \emptyset),$$

tức là một DAG rỗng.

Vì đồ thị phụ thuộc G là DAG không có chu trình, luôn tồn tại một thứ tự topo để sắp xếp các token sao cho chúng có thể di chuyển tuần tự mà không xảy ra xung đột về tính độc lập. Trong ví dụ này, mọi thứ tự topo đều hợp lệ. Ta chọn thứ tự $\pi(1) = a, \pi(2) = e$, nghĩa là token tại a sẽ di chuyển trước token tại e trong các bước tiếp theo của thuật toán (C3–C4).

Việc có được đồ thị phụ thuộc rõ ràng và DAG đảm bảo cho phép chuyển từ quá trình kiểm tra sang giai đoạn dựng chuỗi tái cấu hình một cách an toàn và nhất quán.

C3 - Sắp xếp topo

Sau khi hoàn thành bước C2, ta thu được đồ thị phụ thuộc token là $G = (\{a, e\}, \emptyset)$, tức là một DAG rỗng không chứa cung nào. Điều này có nghĩa là không tồn tại

bất kỳ quan hệ cản nào giữa các token, và vì thế mọi sắp xếp topo của tập token đều hợp lệ.

Tuy nhiên, để tiến hành bước tiếp theo trong quá trình tái cấu hình, ta vẫn cần lựa chọn một thứ tự cụ thể và kiểm tra xem trạng thái trung gian tại mỗi bước di chuyển có duy trì được tính độc lập hay không. Dưới đây là hai phương án sắp xếp topo hợp lệ:

Phương án	**Thứ tự π**	**Trạng thái trung gian sau bước 1**	**Có vi phạm độc lập?**
A	$\pi(1) = a,\ \pi(2) = e$	$\{b, e\}$	Không – b chỉ kề a, c
B	$\pi(1) = e,\ \pi(2) = a$	$\{a, d\}$	Không – d chỉ kề c, e

Bảng 5.5: So sánh hai phương án sắp xếp topo và trạng thái trung gian

Trong phương án A, token tại a di chuyển trước theo đường đi $a \to b$, dẫn đến trạng thái trung gian $\{b, e\}$. Vì b chỉ kề a và c, còn e không kề b, nên tập vẫn giữ được tính độc lập. Tương tự, ở phương án B, token tại e di chuyển trước theo đường đi $e \to d$, cho trạng thái trung gian $\{a, d\}$. Ở đây, d chỉ kề c và e, không kề a, do đó tập vẫn độc lập.

Cả hai phương án đều an toàn. Tuy nhiên, để có sự nhất quán trong triển khai, ta áp dụng một nguyên tắc thường dùng là *ưu tiên token có đường đi ngắn hơn được di chuyển trước*. Trong ví dụ này, cả hai đường $a \to b$ và $e \to d$ đều dài 1 bước, nên không có sự ưu tiên rõ ràng. Ta chọn tuỳ ý và giả sử giữ phương án A: $\pi(1) = a,\ \pi(2) = e$. Lựa chọn này sẽ tiếp tục được xác nhận ở bước C4 khi các token được di chuyển tuần tự theo thứ tự đã định và mỗi cấu hình trung gian đều được kiểm tra về tính độc lập.

Mục đích của bước C3 là chuyển đồ thị phụ thuộc token G thành một thứ tự di chuyển sao cho mỗi token chỉ được trượt khi tất cả token mà nó phụ thuộc đã di chuyển xong, và mọi cấu hình trung gian đều duy trì tính độc lập. Trong ví dụ này, G là DAG rỗng nên điều kiện phụ thuộc được thoả mãn một cách hiển nhiên. Ta chỉ cần kiểm tra nhanh các phương án sắp xếp topo để đảm bảo không vi phạm tính độc lập, sau đó chọn một thứ tự cụ thể.

C4 - Di chuyển tuần tự theo π

Tại bước C4, ta thực hiện việc di chuyển token tuần tự theo thứ tự topo đã cố định từ bước C3, cụ thể là $\pi = (a, e)$, tức token tại a di chuyển trước rồi đến token tại e. Đồ thị ban đầu có 4 cung, và tập ban đầu là $I^s = \{a, e\}$, đích là $I^t = \{b, d\}$.

Trong bước 1, token tại a trượt theo cung $a \to b$, tạo cấu hình trung gian $\{b, e\}$.

Kiểm tra trên đồ thị vô hướng nền cho thấy không có cặp nào trong $\{b, e\}$ kề nhau $\Rightarrow$ vẫn là tập độc lập.

Ở bước 2, token tại e trượt sang d, cho cấu hình $\{b, d\} = I^t$; tại thời điểm này, e đã rỗng nên $\{b, d\}$ cũng là tập độc lập.

Như vậy, chuỗi tái cấu hình hợp lệ là:

$$\{a, e\} \to a \to b \Rightarrow \{b, e\} \to e \to d \Rightarrow \{b, d\}$$

với độ dài đúng bằng tổng trọng số $w(a \to b) + w(e \to d) = 2$, khớp **Hệ quả 4.3.**

Ở mỗi bước, chỉ một token di chuyển, token còn lại không bao giờ kề với token đang trượt, nhờ vậy điều kiện độc lập được duy trì toàn cục.

Ví dụ này minh họa trọn vẹn sự phối hợp giữa hai giai đoạn của thuật toán – Decision khẳng định tồn tại DPM, và Construction thực hiện cụ thể chuỗi tái cấu hình ngắn nhất theo cấu trúc đã xác lập.

5.3.4 No-instance trên đồ thị ngôi sao có hướng

Ví dụ dưới đây minh họa một trường hợp không tồn tại dãy Directed Token Sliding (DTS) giữa I^s và I^t. Ví dụ này đồng thời cho thấy vì sao thuật toán quyết định (mục 4.4.1) dừng lại ở bước B4 và trả lời No.

Xét đồ thị có hướng $T = (V, A)$ với:

- Tập đỉnh:

$$V = \{o,\ \text{in}_1,\ \text{in}_2,\ \text{out}_1,\ \text{out}_2\}$$

- Tập cung:

$$A = \{\text{in}_1 \to o,\ \text{in}_2 \to o,\ o \to \text{out}_1,\ o \to \text{out}_2\}$$

Cấu trúc này tạo thành một *đồ thị ngôi sao bốn lá có hướng*:

- Hai đỉnh in_1, in_2 có cung hướng vào trung tâm o.
- Hai đỉnh $\text{out}_1, \text{out}_2$ nhận cung đi ra từ o.

Tập độc lập ban đầu được ký hiệu là $I^s = \{\text{out}_1, \text{out}_2\}$, tức là tại thời điểm xuất phát, hai token đang được đặt ở hai đỉnh lá ra của cây. Mục tiêu của quá trình tái cấu hình là đưa hai token này đến tập đích $I^t = \{\text{in}_1, \text{in}_2\}$, tức là di chuyển chúng về đúng vị trí của hai lá vào trong cây, theo đúng quy tắc trượt token cùng chiều cung.

I^s và I^t có cùng kích thước 2. Ta dễ dàng nhận thấy rằng không có cạnh nào nối giữa hai đỉnh trong cùng một tập từ đó suy ra cả hai đều là *tập độc lập hợp lệ*. $\Rightarrow$

ví dụ này thỏa mãn điều kiện đầu vào của bài toán Directed Token Sliding.

Bước	Nội dung thao tác	Kết quả & giải thích
B0 – Tiền xử lý	Kiểm tra $\|I^s\| = \|I^t\| = 2$ và không có cặp đỉnh nào kề nhau trong I^s, I^t.	Điều kiện sơ bộ *thỏa* $\Rightarrow$ tiếp tục.
B1 – Tính trọng số	Với mỗi cung $e = (u, v)$, tính $w(e; I^s, I^t) = \|C_e^- \cap I^s\| - \|C_e^- \cap I^t\|$ theo công thức (4.1).	Một số cung có $w(e) < 0$ (sẽ thể hiện chi tiết trong bảng bên dưới).
B2 – Rigid / Blocking	Tìm các rigid token R (mọi cung kề có $w = 0$) và các blocking arc B (một cung có $w = 1$, còn lại $w = 0$).	Không đỉnh/cung nào thỏa điều kiện $\Rightarrow R = B = \varnothing$.
B3 – Tạo T'	Loại bỏ mọi cung kề đỉnh trong R và kề đầu mút cung trong B.	Vì R, B đều rỗng nên $T' = T$.
B4 – Kiểm tra trọng số âm	Nếu tồn tại bất kỳ cung nào có $w(e) < 0$ thì kết luận `NO`.	Có ít nhất một cung có $w(e) < 0 \Rightarrow$ thuật toán dừng và trả lời `NO`.

Bảng 5.6: Các bước xử lý *instance* trong bài toán Directed Token Sliding

Trong trường hợp này, mỗi token xuất phát từ một đỉnh lá phía ngoài (ký hiệu là out_1, out_2) và cần di chuyển đến một đỉnh lá khác ở phía trong (ký hiệu là in_1, in_2). Tuy nhiên, để đến được đích, các token buộc phải di chuyển ngược lại chiều mũi tên trung tâm, tức là đi từ $\text{in}_i \to o$, trong khi quy tắc của bài toán *Directed Token Sliding* chỉ cho phép trượt token theo đúng chiều của cung. Điều này làm cho việc tái cấu hình trở nên bất khả thi. Trọng số âm $w(e) < 0$ trên các cung $\text{in}_i \to o$ phản ánh chính xác sự bất cân bằng này: phía đuôi (tức nguồn token) có ít hơn phía đầu (tức đích cần đến), nghĩa là dòng chảy cần "đi ngược chiều" để bù đắp – điều vốn bị cấm trong mô hình DTS. Đây là biểu hiện rõ ràng của một *instance* không khả thi.

Cung e	**Thành phần** C_e^- **sau khi xóa** e	$\|C_e^- \cap I^s\|$	$\|C_e^- \cap I^t\|$	$w(e)$
$\text{in}_1 \to o$	$\{\text{in}_1\}$	0	1	-1
$\text{in}_2 \to o$	$\{\text{in}_2\}$	0	1	-1
$o \to \text{out}_1$	$\{o,\ \text{in}_1,\ \text{in}_2,\ \text{out}_2\}$	2	2	0
$o \to \text{out}_2$	$\{o,\ \text{in}_1,\ \text{in}_2,\ \text{out}_1\}$	2	2	0

Bảng 5.7: Bảng tính trọng số $w(e)$ cho các cung trong đồ thị ngôi sao bốn lá

Trường hợp ngôi sao bốn lá ta đang xét cho thấy rằng bước B4 đủ mạnh để dừng sớm: chỉ cần một cung âm là khẳng định *no-instance*. Nhờ đó, thuật toán đạt độ phức tạp $O(|V|)$ ngay cả khi không cần xét sâu tới các bước C1→C5.

KẾT LUẬN VÀ CÁC HƯỚNG NGHIÊN CỨU TRONG TƯƠNG LAI

Khóa luận này tập trung nghiên cứu và phân tích sâu về bài toán Directed Token Sliding (DTS) – một biến thể có hướng của bài toán trượt token trên đồ thị, vốn là một trong những bài toán tiêu biểu trong lĩnh vực tái cấu hình (reconfiguration problems). Khác với các mô hình truyền thống trên đồ thị vô hướng, bài toán DTS đặt ra những thách thức mới về mặt thuật toán do đặc trưng bất đối xứng của các cung có hướng. Khóa luận này đã tập trung vào ba nội dung chính:

1. Phân tích độ phức tạp của bài toán DTS trên các lớp đồ thị cụ thể. Khóa luận đã trình bày và giải thích chi tiết hai kết quả quan trọng từ nghiên cứu gần đây:
   - Bài toán PSPACE-complete trên oriented graphs (đồ thị có hướng không có cặp cung ngược chiều).
   - Bài toán NP-complete trên DAGs (đồ thị có hướng không chu trình), thông qua phép giảm từ bài toán Multicolored Independent Set.
2. Xây dựng và trình bày thuật toán quyết định (Decision Algorithm) trong trường hợp đồ thị đầu vào là polytree – lớp đồ thị đặc biệt có đồ thị nền là cây. Với trường hợp này, khóa luận đã:
   - Trình bày rõ ràng từng bước của thuật toán tuyến tính kiểm tra khả năng tái cấu hình *(YES/NO)*.
   - Phân tích chính xác điều kiện cần và đủ thông qua hàm trọng số $w(e)$ trên các cung.
3. Mô tả thuật toán xây dựng chuỗi tái cấu hình trong thời gian nếu là *yes-instance* thuật toán sử dụng kỹ thuật ghép các đường đi có hướng (DPM), xây dựng đồ thị phụ thuộc token và thực hiện sắp xếp topo để đảm bảo di chuyển hợp lệ, giữ nguyên tính chất độc lập tại mỗi bước.

Ngoài ra, khóa luận còn bổ sung nhiều ví dụ minh họa và sơ đồ trực quan giúp làm sáng tỏ các phép quy giảm và logic thuật toán.

Trong khuôn khổ của khóa luận, tôi mới chỉ dừng lại ở việc phân tích lý thuyết và mô phỏng bằng các ví dụ thủ công. Một hướng phát triển quan trọng và thiết thực trong tương lai là hiện thực hóa toàn bộ thuật toán *Directed Token Sliding (DTS)* trên polytree bằng lập trình, với các mục tiêu cụ thể như sau:

- Xây dựng thư viện Python cài đặt các thuật toán:
  - Thuật toán kiểm tra khả năng tái cấu hình (*Decision Algorithm*).
  - Thuật toán xây dựng chuỗi tái cấu hình (*Construction Algorithm*).
  - Tích hợp công cụ trực quan hóa chuyển động token trên đồ thị.
- Phát triển công cụ mô phỏng trực quan, cho phép người dùng:
  - Nhập vào một đồ thị và hai tập token ban đầu – đích.
  - Hệ thống tự động kiểm tra khả năng tái cấu hình và hiển thị toàn bộ quá trình di chuyển token qua từng bước.
- Thực hiện kiểm thử trên các tập dữ liệu thực tế hoặc sinh ngẫu nhiên nhằm:
  - Đánh giá hiệu năng thuật toán trong các trường hợp lớn.
  - Tối ưu hóa về thời gian xử lý và sử dụng bộ nhớ.
- Mở rộng phần mềm để xử lý các lớp đồ thị có hướng phức tạp hơn, như:
  - DAGs, đa cây (*multi-trees*), hoặc đồ thị có trọng số.
  - Qua đó kiểm tra tính tổng quát, độ ổn định và hiệu lực thực tiễn của mô hình.

Việc phát triển phần mềm này không chỉ giúp khóa luận vượt ra ngoài giới hạn lý thuyết, mà còn mở ra tiềm năng ứng dụng trong các lĩnh vực thực tế như: lập kế hoạch chuyển động cho robot, mô phỏng luồng truyền thông một chiều, hoặc thiết kế các hệ thống tương tác động dựa trên cấu trúc đồ thị có hướng.

Tài liệu tham khảo

[1] T. Ito, Y. Iwamasa, Y. Kobayashi, Y. Nakahata, Y. Otachi, M. Takahashi, and K. Wasa, "Independent set reconfiguration on directed graphs," in *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, vol. 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 58:1–58:15, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2022.

[2] VNOI, "Lý thuyết đồ thị - vnoi wiki." `https://wiki.vnoi.info/algo/graph-theory/graph`, n.d. Truy cập: ngày 13/5/2025.

[3] C. H. Papadimitriou, *Computational Complexity*, ch. Some PSPACE Problems, pp. 261–280. Reading, MA: Addison-Wesley, 1994.

[4] M. Sipser, *Introduction to the Theory of Computation*, ch. 8, pp. 347–350. Boston, MA: Cengage Learning, 2 ed., 2006.

[5] L. Stockmeyer, "The polynomial-time hierarchy," *Theoretical Computer Science*, vol. 1, no. 1, pp. 1–22, 1973.

[6] W. J. Savitch, "Relationships between nondeterministic and deterministic tape complexities," *Journal of Computer and System Sciences*, vol. 4, no. 2, pp. 177–192, 1970.

[7] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, (New York, NY, USA), pp. 151–158, ACM, 1971.

[8] R. A. Hearn and E. D. Demaine, "Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation," *Theoretical Computer Science*, vol. 343, no. 1-2, pp. 72–96, 2005.

[9] M. Kaminski, P. Medvedev, and M. Milanic, "Complexity of independent set reconfigurability problems," *Theoretical Computer Science*, vol. 439, pp. 9–15, 2012.

[10] P. S. Bonsma, M. Kaminski, and M. Wrochna, "Reconfiguring independent sets in claw-free graphs," in *Proceedings of the 14th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2014)*, vol. 8503 of *LNCS*, pp. 86–97, 2014.

[11] E. D. Demaine, M. L. Demaine, E. Fox-Epstein, D. A. Hoang, T. Ito, H. Ono, Y. Otachi, R. Uehara, and T. Yamada, "Linear-time algorithm for sliding tokens on trees," *Theoretical Computer Science*, vol. 600, pp. 132–142, 2015.

[12] E. Fox-Epstein, D. A. Hoang, Y. Otachi, and R. Uehara, "Sliding token on bipartite permutation graphs," in *Proceedings of the 26th International Symposium on Algorithms and Computation (ISAAC 2015)*, vol. 9472 of *LNCS*, pp. 237–247, 2015.

[13] D. A. Hoang and R. Uehara, "Sliding tokens on a cactus," in *Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC 2016)*, vol. 64 of *LIPIcs*, pp. 37:1–37:26, 2016.

[14] M. Bonamy and N. Bousquet, "Token sliding on chordal graphs," in *Proceedings of the 43rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2017)*, vol. 10520 of *LNCS*, pp. 127–139, 2017.

[15] N. Bousquet, A. Mary, and A. Parreau, "Token jumping in minor-closed classes," in *Proceedings of the 21st International Symposium on Fundamentals of Computation Theory (FCT 2017)*, vol. 10472 of *LNCS*, pp. 136–149, 2017.

[16] D. Lokshtanov and A. E. Mouawad, "The complexity of independent set reconfiguration on bipartite graphs," *ACM Transactions on Algorithms*, vol. 15, no. 1, pp. 7:1–7:19, 2019.

[17] R. Belmonte, E. J. Kim, M. Lampis, V. Mitsou, Y. Otachi, and F. Sikora, "Token sliding on split graphs," *Theory of Computing Systems*, vol. 65, pp. 662–686, 2021.

[18] T. Ito, M. J. Kaminski, H. Ono, A. Suzuki, R. Uehara, and K. Yamanaka, "Parameterized complexity of independent set reconfiguration problems," *Discrete Applied Mathematics*, vol. 283, pp. 336–345, 2020.

[19] A. Suzuki, A. E. Mouawad, and N. Nishimura, “Reconfiguration of dominating sets,” *Journal of Combinatorial Optimization*, vol. 32, no. 4, pp. 1182–1195, 2016.

[20] A. Haddadan, T. Ito, A. E. Mouawad, N. Nishimura, H. Ono, A. Suzuki, and Y. Tebbal, “The complexity of dominating set reconfiguration,” *Theoretical Computer Science*, vol. 651, pp. 37–49, 2016.

[21] D. Lokshtanov, A. E. Mouawad, F. Panolan, M. S. Ramanujan, and S. Saurabh, “Reconfiguration on sparse graphs,” *Journal of Computer and System Sciences*, vol. 95, pp. 122–131, 2018.

[22] T. Ito, H. Ono, and Y. Otachi, “Reconfiguration of cliques in a graph,” in *Proceedings of the 12th Annual Conference on Theory and Applications of Models of Computation (TAMC 2015)*, vol. 9076 of *LNCS*, pp. 212–223, 2015.

[23] N. Bousquet, T. Hatanaka, T. Ito, and M. M"uhlenthaler, “Shortest reconfiguration of matchings,” in *Proceedings of the 45th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2019)*, vol. 11789 of *LNCS*, pp. 162–174, 2019.

[24] M. Bonamy, N. Bousquet, M. Heinrich, T. Ito, Y. Kobayashi, A. Mary, M. M"uhlenthaler, and K. Wasa, “The perfect matching reconfiguration problem,” in *Proceedings of the 44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*, vol. 138 of *LIPIcs*, pp. 80:1–80:14, 2019.

[25] T. Ito, N. Kakimura, N. Kamiyama, Y. Kobayashi, and Y. Okamoto, “Shortest reconfiguration of perfect matchings via alternating cycles,” in *Proceedings of the 27th Annual European Symposium on Algorithms (ESA 2019)*, vol. 144 of *LIPIcs*, pp. 61:1–61:15, 2019.

[26] P. S. Bonsma and L. Cereceda, “Finding paths between graph colourings: Pspace-completeness and superpolynomial distances,” *Theoretical Computer Science*, vol. 410, pp. 5215–5226, 2009.

[27] H. Osawa, A. Suzuki, T. Ito, and X. Zhou, “Algorithms for coloring reconfiguration under recolorability constraints,” in *Proceedings of the 29th*

International Symposium on Algorithms and Computation (ISAAC 2018), vol. 123 of *LIPIcs*, pp. 37:1–37:13, 2018.

[28] M. Bonamy, M. Heinrich, T. Ito, Y. Kobayashi, H. Mizuta, M. M"uhlenthaler, A. Suzuki, and K. Wasa, "Diameter of colorings under kempe changes," *Theoretical Computer Science*, vol. 838, pp. 45–57, 2020.

[29] J. van den Heuvel, "The complexity of change," in *Surveys in Combinatorics 2013*, vol. 409 of *London Mathematical Society Lecture Note Series*, pp. 127–160, Cambridge University Press, 2013.

[30] N. Nishimura, "Introduction to reconfiguration," *Algorithms*, vol. 11, no. 4, p. 52, 2018.

[31] T. Ito, E. D. Demaine, N. J. A. Harvey, C. H. Papadimitriou, M. Sideri, R. Uehara, and Y. Uno, "On the complexity of reconfiguration problems," *Theoretical Computer Science*, vol. 412, no. 12-14, pp. 1054–1065, 2011.

[32] T. Ito, M. Kaminski, H. Ono, A. Suzuki, R. Uehara, and K. Yamanaka, "On the parameterized complexity for token jumping on graphs," in *Proceedings of the 11th Annual Conference on Theory and Applications of Models of Computation (TAMC 2014)*, vol. 8402 of *LNCS*, pp. 341–351, 2014.

[33] T. Ito, M. J. Kaminski, and H. Ono, "Fixed-parameter tractability of token jumping on planar graphs," in *Proceedings of the 25th International Symposium on Algorithms and Computation (ISAAC 2014)*, vol. 8889 of *LNCS*, pp. 208–219, 2014.

[34] K. Sugimori, "A polynomial-time algorithm for shortest reconfiguration of sliding tokens on a tree," Master's thesis, The University of Tokyo, 2019. In Japanese.

[35] C. Greene and T. Zaslavsky, "On the interpretation of whitney numbers through arrangements of hyperplanes, zonotopes, non-radon partitions, and orientations of graphs," *Transactions of the American Mathematical Society*, vol. 280, no. 1, pp. 97–126, 1983.

[36] K. Fukuda, A. Prodon, and T. Sakuma, "Notes on acyclic orientations and the shelling lemma," *Theoretical Computer Science*, vol. 263, no. 1-2, pp. 9–16, 2001.

[37] T. Ito, Y. Iwamasa, N. Kakimura, N. Kamiyama, Y. Kobayashi, S. ichi Maezawa, Y. Nozaki, Y. Okamoto, and K. Ozeki, "Monotone edge flips to an orientation of maximum edge-connectivity 'a la nash-williams," in *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA 2022)*, pp. 1342–1355, 2022.

[38] O. Aichholzer, J. Cardinal, T. Huynh, K. Knauer, T. M"utze, R. Steiner, and B. Vogtenhuber, "Flip distances between graph orientations," *Algorithmica*, vol. 83, pp. 116–143, 2021.

[39] T. Ito, Y. Iwamasa, Y. Kobayashi, Y. Nakahata, Y. Otachi, and K. Wasa, "Reconfiguring directed trees in a digraph," in *Proceedings of the 27th International Computing and Combinatorics Conference (COCOON 2021)*, vol. 13025 of *LNCS*, pp. 343–354, 2021.

[40] R. A. Hearn and E. D. Demaine, "Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation," *Theoretical Computer Science*, vol. 343, no. 1-2, pp. 72–96, 2005.

[41] R. Aharoni, E. Berger, D. Kotlar, and R. Ziv, "On a conjecture of stein," *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, vol. 87, no. 1, pp. 85–100, 2017.