ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Dư Phương Hạnh

NÂNG CAO HIỆU NĂNG THI HÀNH CÁC PHÉP TOÁN TRÊN ĐỒ THỊ

LUẬN ÁN TIẾN SỸ HỆ THỐNG THÔNG TIN

Hà Nội - 2019

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Dư Phương Hạnh

NÂNG CAO HIỆU NĂNG THI HÀNH CÁC PHÉP TOÁN TRÊN ĐỒ THỊ

LUẬN ÁN TIẾN SỸ HỆ THỐNG THÔNG TIN

Chuyên ngành: Hệ thống thông tin
Mã số: 9480104.01

Người hướng dẫn khoa học:
1. PGS.TS. Nguyễn Hải Châu
2. PGS.TS. Nguyễn Kim Khoa

Hà Nội - 2019

LỜI CẢM ƠN

Luận án được thực hiện tại Trường Đại học Công nghệ - ĐHGQ Hà Nội, dưới sự hướng dẫn của PGS.TS. Nguyễn Hải Châu và PGS.TS. Nguyễn Kim Khoa (Trường ETS - Đại học Quebec - Canada).

Trước hết, tôi xin được bày tỏ lòng biết ơn sâu sắc tới PGS.TS. Nguyễn Hải Châu và PGS.TS. Nguyễn Kim Khoa, những người đã hướng dẫn, đưa ra những định hướng giúp tác giả hoàn thành bản luận án này. Tôi cũng chân thành cám ơn toàn thể các thầy, cô đồng nghiệp trong Bộ môn Các hệ thống thông tin đã đồng hành trong nhiều năm, góp nhiều ý kiến quan trọng để tác giả có thể hoàn thiện các nội dung khoa học của luận án.

Để có được kết quả ngày hôm nay, tôi cũng xin chân thành cảm ơn Trường Đại học Công nghệ, Khoa Công nghệ Thông tin, các Phòng chức năng của Trường, đã tạo điều kiện thuận lợi cho tôi trong quá trình nghiên cứu và công tác tại Trường.

Sau cùng, tôi xin chân thành cảm ơn gia đình, những người thân và bạn bè đã giúp đỡ, động viên tôi trong suốt thời gian thực hiện luận án này!

Hà Nội, tháng 08 năm 2019

Dư Phương Hạnh

LỜI CAM ĐOAN

Tôi cam đoan đây là công trình nghiên cứu của riêng tôi. Các nội dung viết chung với các tác giả khác đều được sự đồng ý của đồng tác giả trước khi đưa vào Luận án. Các kết quả nêu trong luận án là trung thực và chưa từng được ai công bố trong các công trình nào khác.

Tác giả

Dư Phương Hạnh

Mục lục

Danh sách hình vẽ

Danh sách bảng

Danh sách thuật toán

Danh mục thuật ngữ viết tắt

BFS	Breadth-First Search	Giải thuật duyệt theo chiều rộng trước
bBFS	binary BFS	Giải thuật duyệt BFS theo hai chiều
DFS	Depth-First Search	Giải thuật duyệt theo chiều sâu trước
SSSP	Single Source Shortest Path	Bài toán tìm đường đi ngắn nhất từ một đỉnh đến tất cả các đỉnh còn lại trong đồ thị
APSP	All Pairs Shortest Path	Bài toán tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh trong đồ thị
CC	Closeness Centrality	Độ trung tâm gần
BC	Betweenness Centrality	Độ trung tâm trung gian
CSDL		Cơ sở dữ liệu

Chương 1

GIỚI THIỆU CHUNG

1.1 Động lực nghiên cứu

Lý thyết đồ thị đang được ứng dụng rộng rãi hiện nay, đặc biệt khi lượng dữ liệu chúng ta cần phải xử lý, phân tích để trích rút tri thức có quy mô ngày càng lớn. Dữ liệu hiện được xem như nguồn tài sản quý giá cho mỗi tổ chức, cá nhân, thậm chí được ví như "vàng"[71]. Trong định hướng phát triển nguồn kinh tế số, dữ liệu lại càng trở nên quan trọng, quyết định đến sự thành công hay thất bại của mỗi cá nhân, tổ chức. Với sự dịch chuyển sang nền kinh tế số, lượng dữ liệu rõ ràng ngày càng được sinh ra nhiều hơn, đồ sộ hơn về cả quy mô, tốc độ, tính đa dạng và tính chân thật của dữ liệu. Theo thống kê của tổ chức Internet Live Stats đến tháng 09 năm 2019, tốc độ vận chuyển dữ liệu qua Internet khoảng 79.870 GB/giây; 4,8 tỷ truy vấn tìm kiếm Google mỗi ngày; 184 tỷ email được gửi đi mỗi ngày; ... [66]. Sự đa dạng về loại và quy mô dữ liệu đã dẫn đến những mô hình dữ liệu truyền thống như mô hình quan hệ gặp khó khăn khi xử lý [29]. Với hiện trạng đó, một số mô hình quản lý dữ liệu mới đã được đề xuất. Hai trong số những mô hình dữ liệu hiện được xem là hiệu quả đối khi quản lý dữ liệu quy mô lớn hiện vẫn là mô hình dữ liệu hướng tài liệu (document-based model) và mô hình dữ liệu đồ thị (graph data model) [30].

Việc áp dụng lý thuyết đồ thị vào các bài toán thực tiễn đã được tiến hành từ lâu. Tuy nhiên, khi lượng dữ liệu ngày càng lớn, chẳng hạn dữ liệu từ các mạng xã hội như Facebook, thì việc mô hình hoá dữ liệu bằng lý thuyết đồ thị lại được quan tâm và đã minh chứng được hiệu năng nổi bật khi áp dụng vào thực tế [75]. Khi mô hình hoá dữ liệu bằng đồ thị, thông thường các thực thể (chẳng hạn các thành viên mạng xã hội) sẽ được biểu diễn thông qua các đỉnh còn các quan hệ giữa các thực thể (chẳng hạn như quan hệ bạn bè giữa các thành viên) được quy về các cạnh liên kết các đỉnh trong đồ thị [53].

Đối với các bài toán mô hình hoá bằng đồ thị có quy mô lớn về cả số đỉnh và số cạnh, một trong những thách thức lớn được đặt ra là cần phải có (i) những phương pháp tổ chức dữ liệu đồ thị hiệu quả và phương pháp nâng cao hiệu năng các phép toán phân tích đồ, bao hàm cả (ii) tối ưu hoá các truy vấn khoảng cách ngắn nhất trên đồ thị động và (iii) cải

thiện hiệu năng tính toán một số độ đo quan trọng phục vụ các phép phân tích đồ thị quy mô lớn.

1.1.1 Cấu trúc dữ liệu phù hợp để nâng cao hiệu năng thi hành các phép toán trên đồ thị

Với cách tiếp cận sử dụng lý thuyết đồ thị để giải những bài toán thực tế, việc lựa chọn và xác định cấu trúc dữ liệu để biểu diễn đồ thị quyết định trực tiếp đến việc hình thành giải pháp cũng như hiệu năng của các phép toán trên đồ thị. Chẳng hạn như với các mạng xã hội như Facebook, Twitter, ..., chúng ta cần phải lựa chọn được phương pháp biểu diễn dữ liệu liên quan đến các thành viên và những quan hệ giữa các thành viên trong mạng một cách hiệu quả thì mới có thể triển khai thực tế được với số lượng từ vài trăm triệu đến hàng tỷ thành viên.

Trong lý thuyết đồ thị, việc tổ chức dữ liệu đồ thị $G = (V, E)$ với V là tập đỉnh và E là tập cạnh, thì mỗi đỉnh của V thông thường được định danh bởi một số tự nhiên. Tập các cạnh E thông thường có thể sử dụng các phương pháp biểu diễn dữ liệu như danh sách cạnh; ma trận liền kề; danh sách liền kề; ma trận liên thuộc; ...[108]. Tuy nhiên, hiện nay, phương pháp biểu diễn dựa theo danh sách liền kề là cách tiếp cận phù hợp nhất, đặc biệt khi đồ thị có quy mô lớn về số đỉnh, số cạnh [102].

Các nghiên cứu hiện tại trong việc xây dựng phương pháp biểu diễn đồ thị hiện nay đều chưa quan tâm nhiều đến tính cục bộ dữ liệu để khai thác vai trò và chức năng của bộ nhớ đệm (cache) trong các hệ thống tính toán [90]. Theo nghiên cứu trong công bố [114], các công cụ, thư viện phân tích đồ thị hiện nay thường thực hiện rất ít tính toán cho mỗi dữ liệu đồ thị được truy cập trong khi phần lớn truy cập bộ nhớ chính (main memory) lại là ngẫu nhiên. Điều đó dẫn đến tỷ lệ dữ liệu đồ thị cần phải đưa vào và thay thế trong bộ nhớ cache cao (cache miss), làm giảm hiệu năng phân tích đồ thị [78]. Đây là một trong những động lực hình thành nên bài toán nghiên cứu của luận án này.

1.1.2 Xử lý các truy vấn khoảng cách ngắn nhất trên đồ thị động quy mô lớn

Ngày nay, các mạng xã hội đóng một vai trò quan trọng trong "xã hội kết nối mạng" của chúng ta. Facebook, Twitter, WhatsApp, ..., được sử dụng phổ biến trong cuộc sống hàng ngày. Để mô hình hóa các mạng xã hội bằng lý thuyết đồ thị, mỗi thành viên thường được mô hình hóa bởi một đỉnh, và mối quan hệ trực tiếp giữa hai thành viên được đại diện bởi một cạnh. Đối với các bài toán liên quan đến phân tích mạng xã hội, có ba vấn đến lớn cần phải xem xét là: (i) số lượng đỉnh và cạnh rất lớn; (ii) đồ thị có tính động do sự thay đổi mối quan hệ giữa các thành viên và thành viên mới đã đăng ký; và (iii) thường xuyên sử dụng truy vấn khoảng cách ngắn nhất để tìm cách xác lập quan hệ giữa hai thành viên [39].

Các vấn đề này định hình nên bài toán cần phải xử lý hiệu quả các truy vấn khoảng cách ngắn nhất (chẳng hạn vừa cập nhật đồ thị vừa truy vấn khoảng cách) trên đồ thị động quy mô lớn.

Nhìn chung, các phép duyệt đồ thị để xác định khoảng cách ngắn nhất giữa hai đỉnh phục vụ cho nhiều phép toán phân tích đồ thị, mạng xã hội. Chẳng hạn như phân tích ảnh hưởng của một người dùng đối với cộng đồng thành viên [110]; để xác định sự gần gũi giữa hai người dùng; để tìm thêm người dùng hoặc nội dung liên quan bằng cách sử dụng tìm kiếm trên mạng [39]. Mặc dù bài toán xử lý truy vấn xác định khoảng cách ngắn nhất đã rất kinh điển, tuy nhiên việc thi hành các truy vấn này một cách tối ưu trên một đồ thị, một mạng xã hội vừa có quy mô lớn, vừa có sự cập nhật thay đổi liên tục về số thành viên lẫn quan hệ lại là một thách thức lớn trong thực tế [104].

1.1.3 Nâng cao hiệu năng tính các độ đo quan trọng trong phân tích đồ thị quy mô lớn

Mạng xã hội ngày nay đã có mặt khắp mọi nơi, trên mọi quốc gia và đã trở thành một phương tiện quan trọng để kết nối mọi người trong một xã hội. Facebook, Twitter, YouTube và WhatsApp là những mạng xã hội rất phổ biến và đã trở nên quan trọng đối với rất nhiều người trong cuộc sống hiện đại của chúng ta. Theo thống kê được cung cấp bởi Cổng thông tin thống kê vào tháng 9 năm 2019, số lượng người dùng hoạt động của Facebook là 2,327 tỷ; Youtube là 1,9 tỷ và WhatsApp đã vượt 1,5 tỷ thành viên [66]. Lý thuyết đồ thị đã được coi là một phương pháp thích hợp để mô hình hóa các mạng xã hội. Một thành viên của một mạng xã hội thường được mô hình hóa bởi một đỉnh, và mối quan hệ trực tiếp giữa hai thành viên được đại diện bởi một cạnh. Để quản lý mạng xã hội, nhiều phương pháp phân tích mạng xã hội đã được đề xuất và sử dụng trong thực tế. Phân tích mạng xã hội được định nghĩa là quá trình điều tra các cấu trúc mạng xã hội thông qua việc sử dụng mạng và lý thuyết đồ thị [85]. Vì vậy, phương pháp này được coi là một kỹ thuật quan trọng trong xã hội học hiện đại.

Một trong những điều quan trọng mà chúng ta cần phải tính toán khi thực hiện phân tích mạng là xác định độ trung tâm của một nút trong mạng xã hội. Hay nói cách khác, chúng ta cần phải tính toán, phân tích mạng để có thể xác định được đỉnh (tức thành viên) có ảnh hưởng lớn nhất đến các đỉnh khác [67]. Từ đó có thể thấy độ trung tâm của một đỉnh trong lý thuyết đồ thị cho phép chúng ta xác định những người dùng quan trọng nhất trong một mạng [35].

Hai trong những chỉ dấu trung tâm được sử dụng rộng rãi nhất trong các bài toán phân tích mạng xã hội là *"độ trung tâm gần" (closeness centrality)* và *"độ trung tâm trung gian" (betweenness centrality)* [53]. Các phương pháp hiện nay đang được sử dụng để tính chính xác các độ đo này tuy đều có độ phức tạp đa thức nhưng với quy mô lớn về số đỉnh và cạnh thì thời gian tính toán đều cũng rất lớn [64]. Từ đó đặt ra nhu cầu cần phải có được những

công cụ, thư viện để nâng cao hiệu năng tính toán các độ đo trung tâm, nâng cao hiệu quả bài toán phân tích đồ thị nói chung. Chính vì thế, hai độ đo này cũng sẽ được chúng tôi tập trung nghiên cứu trong luận án với định hướng kết hợp được cả việc tổ chức dữ liệu đồ thị hợp lý lẫn phương pháp tính toán song song hiệu quả.

1.2 Một số nghiên cứu liên quan

Nâng cao hiệu năng xử lý các phép toán trên đồ thị là hướng nghiên cứu được nhiều nhóm, tổ chức nghiên cứu trên thế giới quan tâm hiện nay. Trong phần này, chúng tôi sẽ giới thiệu một số hướng nghiên cứu liên quan đến bài toán đặt ra trong luận án theo ba nhóm chính: (i) một số luận án tiến sỹ; (ii) các công trình khoa học đã được công bố gần đây liên quan đến hướng nghiên cứu của luận án; và (iii) một số nhóm nghiên cứu hiện đang có những hoạt động khoa học liên quan đến hướng nghiên cứu của luận án.
Nâng cao hiệu năng xử lý các truy vấn trên đồ thị cũng đã được nhiều luận án tiến sỹ chú trọng nghiên cứu trong những năm gần đây. Một số luận án điển hình có thể kể gồm:

- Luận án của Slota [98], công bố năm 2016 tại trường Đại học Pennsylvania, Mỹ, có các đóng góp chính là đề xuất giải pháp song song và khả mở để thi hành một số phép phân tích đồ thị, bao gồm cả duyệt theo chiều rộng trước BFS, dựa trên các hệ thống tính toán có kiến trúc cả đa lõi (kể cả sử dụng GPU) với mô hình chia sẻ bộ nhớ và phân tán với mô hình lai kết hợp MPI và OpenMP. Giải pháp của Slota đã có thể thao tác được với những đồ thị quy mô lớn đến hàng tỷ đỉnh và hàng trăm tỷ cạnh. Tuy nhiên, luận án của Slota mới chỉ tập trung vào các phép toán phân tích kiểu chỉ đọc dữ liệu đồ thị là chính mà chưa quan tâm đến việc thi hành các phép toán tương tranh có cập nhật trên đồ thị quy mô lớn.

- Luận án của Beamer năm 2016 tại Đại học California, Berkeley-Mỹ, có mục tiêu nâng cao hiệu năng một số giải thuật phân tích đồ thị trên kiến trúc chia sẻ bộ nhớ. Các kết quả của luận án bao hàm cả về mặt đóng góp về tổ chức dữ liệu đồ thị nhằm nâng cao tính cục bộ dữ liệu để nâng cao tỷ lệ cache hit lẫn đóng góp về tối ưu hướng duyệt BFS trên đồ thị có đường kính nhỏ [11]. Việc cải thiện hướng duyệt BFS của Beamer cũng dựa trên mẹo (heuristic) để chuyển đổi hướng duyệt dựa trên tính ngưỡng đã duyệt của mỗi chiều. Cũng tương tự như luận án của Slota, các phép toán tương tranh bao hàm cả cập nhật đồ thị chưa được quan tâm trong luận án này.

- Để xử lý các phép toán trên đồ thị có quy mô lớn, luận án của Guerrieri (năm 2015 tại trường Đại học Trento, Ý) [43] đã có những đóng góp trong việc đưa ra một số mô hình tính toán song song trên kiến trúc bộ nhớ phân tán. Guerrieri đã đề xuất giải thuật DFEP (Distributed Funding-based Edge Partitioning) để phân mảnh dữ liệu cạnh đồ thị, xây dựng hệ thống xử lý đồ thị theo mô hình phân mảnh (được gọi là ETSCH)

và giải thuật phân cụm phân tán đối với các ứng dụng suy diễn và tránh nhập nhằng từ. Hiện các đóng góp này chưa được triển khai trên các mô hình tính toán phân tán hiện đại như Apache Spark [1] và hiệu năng của các phép toán phân tích đồ thị còn phụ thuộc quá nhiều vào các mảnh dữ liệu, chưa khai thác được tính cục bộ dữ liệu để cải thiện tỷ lệ cache hit [43].

- Năm 2014, Kyrola đã bảo vệ thành công luận án tại trường Đại học CMU - Mỹ, với mục tiêu nghiên cứu đề xuất giải pháp quản lý đồ thị quy mô lớn trên các máy tính PC [56]. Trong luận án này, Kyrola đã có được hai đóng góp rất quan trọng là đề xuất được (i) giải thuật trượt cửa sổ song song (Parallel Sliding Windows algorithm) để tổ chức tập cạnh của đồ thị thành tập các phân mảnh cạnh đồ thị quy mô lớn; (ii) cấu trúc dữ liệu kiểu danh sách liền kề phân mảnh (Partitioned Adjacency Lists) để có thể biểu diễn được toàn bộ dữ liệu đồ thị thuộc tính, ngay cả khi quy mô đến hàng trăm tỷ cạnh. Với các cấu trúc dữ liệu rút gọn đó, các phép toán tương tranh trên đồ thị như BFS, PageRank, tính thành phần liên thông,... trên đồ thị hàng trăm tỷ cạnh đã có thể thi hành trên một máy tính thực nghiệm có cấu hình cao. Khác với hai luận án trên, Kyrola đã xây dựng và công bố hai nền tảng là GraphChi và GraphChi-DB [2] để hỗ trợ xử lý các truy vấn đồng thời trên đồ thị. Với cách tiếp cận hướng đến sử dụng thiết bị lưu trữ ngoài (dạng SSD) để biểu diễn dữ liệu đồ thị quy mô lớn, nền tảng GraphChi-DB chưa khai thác hết được năng lực của hệ thống tính toán có bộ nhớ lớn cũng như số lượng CPU nhiều do còn phụ thuộc nhiều vào thời gian vào/ra [22].

Ngoài các luận án nêu trên, các nghiên cứu liên quan đến nâng cao hiệu năng các phép toán trên đồ thị cũng nhận được sự quan tâm của nhiều tổ chức và các nhà nghiên cứu trên thế giới. Để phục vụ xử lý các thao tác trên đồ thị, đã có tương đối nhiều công cụ và thư viện phần mềm được xây dựng nhằm đáp ứng yêu cầu đó. Trong số các công cụ đó, có thể kể đến *NetworkX*, một gói phần mềm viết bằng ngôn ngữ Python để tạo, thao tác và phân tích các đồ thị, mạng phức tạp [44]. Bộ thư viện *SNAP C++ library* [45] cũng là một trong số những thư viện nổi tiếng trong việc thao tác và phân tích các đồ thị lớn trên các hệ thống tính toán hiệu năng cao. Tuy nhiên, các phần mềm này lại chưa được cài đặt để tối ưu hoá việc xử lý các truy vấn trên đồ thị: chẳng hạn như việc thi hành các truy vấn tìm đường đi ngắn nhất giữa hai đỉnh, mặc dù cũng đã cài đặt giải thuật tìm theo chiều rộng trước theo cả hai chiều (bidirectional BFS - bBFS), nhưng cả hai phần mềm trên đều thi hành giải thuật đó một cách tuần tự (chỉ sử dụng một luồng tính toán trên một lõi CPU). Ngoài ra, việc lựa chọn chiều để duyệt trong bBFS chỉ dựa vào số đỉnh đã đưa vào trong danh sách hàng chờ của mỗi hướng. Điều đó có thể dẫn đến tình huống ở lần duyệt kế tiếp chúng ta sẽ phải đưa vào trong hàng đợi để duyệt quá nhiều đỉnh.

Thách thức đặt ra với các đồ thị có quy mô lớn cũng đã thu hút được sự quan tâm nghiên

[1] Xem thêm thông tin tại `https://spark.apache.org/`

[2] Xem thêm tại trang `https://github.com/GraphChi/`

cứu của nhiều tổ chức. Các nghiên cứu gần đây đã xây dựng được một số các công cụ thao tác với đồ thị quy mô lớn như GraphLab [107], PowerGraph [40], GraphX [41], ... Các công cụ này đều được xây dựng để xử lý trên cả môi trường tính toán song song lẫn phân tán. Nhìn chung, chúng đều được đánh giá có hiệu quả đối với các bài toán tổng quát khi có các hạ tầng tính toán hiệu năng cao như các cụm máy tính clusters hay trên các siêu máy tính supercomputers [107]. Tuy nhiên, chúng lại không phù hợp khi xử lý các truy vấn tương tranh trên các đồ thị có tính thay đổi nhanh (như mạng xã hội) và chỉ được tính toán trên nền tảng trung bình (chẳng như các máy chủ thông thường hay thậm chí các máy tính cá nhân) tương tự như NetworkX và SNAP C++.

Đối với việc xử lý các phép toán trên đồ thị động, bài toán tối ưu quá trình thi hành truy vấn tìm đường đi ngắn nhất một cách hiệu quả cũng thu hút được sự quan tâm của nhiều nhà nghiên cứu. Các công trình [80], [104], [92] thể hiện các kết quả trong việc xử lý các truy vấn tìm đường đi tối ưu trong các đồ thị giao thông động và trực tuyến. Để có thể tận dụng được kiến trúc đa lõi multi-core trong các hệ thống tính toán hiện nay, các công trình [19], [65], [58] và [7] tập trung đến việc đưa ra những giải pháp song song hoá giải thuật BFS trên các đồ thị quy mô lớn. Mặc dù vậy, các phép toán cập nhật trên đồ thị lại chưa được quan tâm trong các công trình đó.

Trong khuôn khổ cuộc thi ACM SigMod Programming Contest được tổ chức đồng hành với hội thảo SigMod năm 2016 tại Mỹ, bài toán tối ưu hoá các truy vấn tính khoảng cách ngắn nhất giữa hai đỉnh trong đồ thị quy mô lớn, động, có sự cập nhật liên tục cũng đã được đặt ra [97]. Cuộc thi này đã có sự tham dự của 33 nhóm nghiên cứu đến từ rất nhiều trường Đại học cũng như các tổ chức nghiên cứu trên thế giới. Chúng tôi cũng đã tham gia cuộc thi này và đã đạt giải ba (được mời đến trình bày tại hội nghị SigMod 2016 tại Mỹ). Ý tưởng chính tối ưu các truy vấn trên đồ thị của 5 nhóm đoạt giải được trình bày dưới đây:

1. **H_minor_free**: đây là nhóm đã đạt giải nhất của cuộc thi này. Ý tưởng chính của nhóm H_minor_free dựa trên nhúng trạng thái các cạnh vào dữ liệu của đỉnh liền kề. Có ba trạng thái đối với mỗi cạnh: ALIVE đồng nghĩa là cạnh đó còn trong G; DEAD có nghĩa cạnh đó đã bị loại bỏ trong G; và UNKNOWN là trạng thái mà cạnh đó vừa được cập nhật (thêm/xoá) nhưng chưa ghi nhận trong G. Với mỗi tập các phép toán tương tranh, H_minor_free tiến hành qua 3 bước: (i) cập nhật các phép toán $'A'$,$'D'$ và thiết lập trạng thái UNKNOWN cho các cạnh đó; (ii) sử dụng giải thuật bBFS trong truy vấn 'Q' để tính khoảng cách ngắn nhất dựa trên một luồng OpenMP; và (iii) ghi nhận trạng thái cuối cùng cho các cạnh đã cập nhật về ALIVE hoặc DEAD [46][3].

2. **uoa_team**: nhóm sử dụng cách tiếp cận cấu trúc dữ liệu đa phiên bản (multiversioning) cho các phép toán cập nhật đồ thị và sử dụng giải thuật mẹo (heuristics) để tối ưu các truy vấn bBFS theo nhiều luồng. Việc xử lý song song các truy vấn 'Q' được giao

[3] `http://dsg.uwaterloo.ca/sigmod16contest/downloads/uoa_team.tar.gz`

phó cho thư viện *threadpool*11[4] và *concurrentqueue*[5]. Nhóm này đạt giải nhì trong cuộc thi này[6].

3. **akGroup**: đây là giải pháp của chúng tôi dựa trên việc tổ chức dữ liệu đồng thời cả danh sách đỉnh đến và đi có kèm chỉ mục. Các phép toán cập nhật sẽ được ghi nhận trước khi tiến hành song song bBFS với việc sử dụng mẹo (heuristic) để lựa chọn chiều duyệt BFS một cách hiệu quả [DPH1]footnote`http://dsg.uwaterloo.ca/sigmod16contest/downloads/akgroup.tar.gz`.

4. **gStreamPKU**: giải pháp của nhóm dựa trên (i) giảm số phép toán cơ bản trong mỗi truy vấn với phương pháp nén và tối ưu bit để tăng tính cục bộ dữ liệu, từ đó tăng tỷ lệ cache hit; và (ii) xây dựng đồ thị Delta song song hoá các phép toán cập nhật để hỗ trợ xử lý các truy vấn $'Q'$ theo lô với thư viện TBB[7] [8].

5. **while1**: nhóm này sử dụng ý tưởng "danh sách cạnh giao tác" (transaction edge list) để thi hành các phép toán cập nhật. Ngoài ra, để song song hoá xử lý truy vấn, tất cả đỉnh và cạnh của G sẽ được nhân bản trên tất cả các nút NUMA (mô hình xử lý phân tán truy cập bộ nhớ không đồng nhất) để đảm bảo tính cục bộ dữ liệu trong bộ nhớ cache khi xử lý song song bBFS[9].

Trong các bài toán phân tích đồ thị/mạng xã hội (chẳng hạn, tìm người có ảnh hưởng nhất trên mạng xã hội; tìm người tạo điều kiện thuận lợi nhất cho việc truyền thông tin trong mạng lưới khủng bố [52]; hay những protein nào là quan trọng nhất trong mạng sinh học [51]), *các độ đo trung tâm* được sử dụng rộng rãi để đo lường tầm quan trọng tương đối của các đỉnh trong đồ thị [37]. Trong số các độ đo trung tâm, độ trung tâm gần là độ trung tâm cho phép xác định được tương quan ảnh hưởng của một đỉnh với các đỉnh còn lại: độ trung tâm gần của đỉnh v càng lớn thì v càng gần với các đỉnh còn lại [14]. Ngoài độ trung tâm gần, độ trung tâm trung gian được Freeman đề xuất năm 1977 [36], cũng là độ đo được sử dụng rộng rãi để tìm những đỉnh quan trọng trong đồ thị. Độ đo này chính là số lượng cầu nối trung gian một đỉnh đảm nhiệm khi xác lập các quan hệ ngắn nhất giữa những đỉnh khác. Hay nói cách khác, đỉnh có xác suất cao nằm trên đường đi ngắn nhất giữa hai đỉnh bất kỳ thì sẽ có độ trung tâm trung gian cao. Độ đo này đã được áp dụng vào phân tích đồ thị/mạng trong nhiều lĩnh vực khác nhau như vận tải [112], sinh học - y tế [16], phân tích mạng xã hội để phát hiện cộng đồng [23], phát hiện nguy cơ khủng bố [53], ...

Thực tế, các độ đo trung tâm ban đầu được áp dụng đối với các đồ thị có số lượng đỉnh/cạnh không quá lớn trong khi các đồ thị/mạng xã hội hiện này của chúng ta có quy

[4] `https://github.com/tghosgor/threadpool11/`
[5] `https://github.com/cameron314/concurrentqueue/`
[6] `https://dsg.uwaterloo.ca/sigmod16contest/downloads/uoa_team.tar.gz`
[7] `https://www.threadingbuildingblocks.org/`
[8] `http://dsg.uwaterloo.ca/sigmod16contest/downloads/gStreamPKU.tar.gz`
[9] `http://dsg.uwaterloo.ca/sigmod16contest/downloads/while1.tar.gz`

mô số đỉnh/cạnh đều rất lớn (vài triệu đến hàng tỷ đỉnh/cạnh) [52]. Điều này cũng dẫn đến việc thi hành tính toán xác định độ đo trung tâm gần của tất cả các đỉnh trong đồ thị với quy mô lớn thường không thể thi hành tuần tự [21].

Việc tính độ đo trung tâm cũng đã được tích hợp vào rất nhiều các bộ thư viện thao tác với đồ thị. Chẳng hạn như bộ thư viện NetworkX, cung cấp các chức năng để tạo, thao tác, phân tích các đồ thị và mạng xã hội [44]. Thư viện SNAP C++ [60] cũng rất phổ biến để tiến hành các phép toán cơ bản trên đồ thị, ... Các công cụ này cũng đều hỗ trợ các hàm để tính một số độ đo trung tâm. Tuy nhiên, chúng không được cài đặt để thi hành song song và dẫn đến hiệu năng tính toán của các thư viện này đều rất chậm.

Song song hoá các giải thuật tính độ đo trung tâm được xem như một trong những phương pháp hiệu quả nhất để có thể cải tiến hiệu năng tính toán độ trung tâm gần. Cách tiếp cận này sẽ khai thác các kiến trúc bộ nhớ chia sẻ/phân tán và đa lõi, đa CPU để thi hành song song các phép toán tính BFS trên các đỉnh khác nhau (tức thi hành bài toán SSSP). Điển hình như công trình của Chakaravarthy và cộng sự [19] tiến hành song song hoá bài toán SSSP trên các hệ thống tính toán hiệu năng cao. NetworKit là thư viện mã mở được xây dựng với mục tiêu hỗ trợ các phép phân tích mạng đồ thị quy mô lớn [99]. Trong thư viện này, phần lõi được xây dựng sử dụng ngôn ngữ C++ nhằm khai thác các thư viện tính toán song song theo mô hình luồng, cụ thể là sử dụng OpenMP để song song hoá các phép toán đồng thời. Tương tự, TeexGraph cũng là bộ thư viện mã mở viết bằng C++ và sử dụng OpenMP để cài đặt các phép toán phân tích mạng xã hội [101]. Tuy nhiên, các công trình này chưa xét và khai thác đến mô hình phân cấp bộ nhớ (tương quan về hiệu năng, dung lượng giữa bộ nhớ chính RAM và bộ nhớ cache của CPU) trong các hệ thống tính toán: nếu chúng ta có được một cấu trúc dữ liệu hợp lý, chúng ta có thể giảm được tỷ lệ *cache miss* và tăng *cache hit* [DPH1].

Ngoài việc sử dụng các mô hình chia sẻ bộ nhớ để tiến hành song song phép tính độ đo trung tâm, một số giải pháp khác cũng đã được đề xuất để thực hiện công việc này trên các hệ thống tính toán hiệu năng cao phân tán. Chẳng hạn, Pregel [69], GraphLab [107] và PowerGraph [40] có thể được xem như những bộ công cụ nổi bật nhắm tới mục tiêu đó. Phiên bản mở rộng sau này của Pregel được cộng đồng Apache tiếp tục phát triển hình thành Giraph [25] đã có thể thao tác với những mạng có đến hàng nghìn tỷ cạnh [24]. Các bộ công cụ này được thiết kế chủ yếu để phân tích các mạng ở quy mô rất lớn và phải sử dụng những hạ tầng tính toán phức tạp như hệ thống máy tính cụm hay siêu máy tính [107]. Từ đó, chúng cũng không thực sự hiệu quả đối với bài toán cần tính toán độ trung tâm với các mạng thực cỡ không quá lớn như Facebook và môi trường tính toán hạn chế.

Để cải thiện tốc độ tính toán độ trung tâm trung gian, nhiều giải pháp tính xấp xỉ độ đo này đã được đề xuất. Chẳng hạn các công trình [32], [20], [89], [68] đã đề xuất ý tưởng tính nhanh xấp xỉ độ trung tâm trung gian dựa trên kỹ thuật lấy mẫu. Ngoài cách tiếp cận tính xấp xỉ, các công trình [6], [88], [91], [12], [105] đã chú trọng xây dựng những kỹ thuật mẹo để cải thiện hiệu năng tính độ trung tâm trung gian dựa trên khai thác cấu trúc topo

đồ thị, chẳng hạn như tính trước đối với các đỉnh bậc 1, phân vùng đồ thị, xét các đỉnh bậc 2 để tái sử dụng lại kết quả hai cây con của đỉnh đó nhằm giảm thời gian tính pha tích luỹ của giải thuật Brandes.

Ngoài các phương pháp nêu trên, để phục vụ cho việc tính nhanh độ đo này đối với những đồ thị có sự cập nhật nhanh, năm 2018 Jamour và các cộng sự đã đề xuất giải thuật iCENTRAL tính độ trung tâm trung gian tăng dần dựa trên việc phân chia đồ thị thành các thành phần song liên thông (biconnected components) [50].

Đối với các đồ thị tĩnh, việc khai thác các bộ xử lý đồ hoạ GPU (Graphic Processing Unit) cũng được quan tâm nghiên cứu trong việc tính toán song song độ trung tâm trung gian. Năm 2016, Bernaschi và cộng sự đã xây dựng giải pháp MGBC (Multi-GPU Betweenness Centrality) kết hợp sử dụng cả nhiều bộ GPU lẫn mô hình song song sử dụng bộ nhớ phân tán MPI để nâng cao tốc độ tính độ trung tâm trung gian [13]. Việc dùng GPU để tính song song độ đo này đa phần đều sử dụng giải thuật kinh điển của Brandes [34], [70].

Một số giải pháp khác cũng đã được đề xuất để phục vụ tính độ trung tâm trung gian trên các hệ thống tính toán hiệu năng cao phân tán. Chẳng hạn, GraphLab [107] hay Apache Giraph [25] đã có thể thao tác với những mạng có đến hàng nghìn tỷ cạnh [24]. Các bộ công cụ này được thiết kế chủ yếu để phân tích các mạng ở quy mô rất lớn và phải sử dụng những hạ tầng tính toán phức tạp như hệ thống máy tính cụm hay siêu máy tính [107]. Cũng như đã nhận xét ở mục trên, với những yêu cầu hạ tầng tính toán như thế, các giải pháp này không thực sự hiệu quả đối với bài toán cần tính toán độ trung tâm với các mạng thực cỡ không quá lớn như Facebook.

Ngoài các công trình nghiên cứu liên quan nêu trên, bài toán đặt ra trong luận án có thể tìm thấy trong các hoạt động của một số tổ chức nghiên cứu lớn như:

- Nhóm nghiên cứu phân tích mạng xã hội (Social Network Analysis Group)[10] của trường Đại học Stanford, Mỹ, với những nghiên cứu chuyên sâu về mô hình hoá, xử lý và phân tích sâu các mạng xã hội điển hình. Đây cũng là nhóm đã cung cấp rất nhiều các bộ dữ liệu đồ thị/mạng xã hội cho cộng đồng nghiên cứu (gọi tắt là SNAP và cũng được luận án sử dụng trong các thực nghiệm).

- Nhóm nghiên cứu hệ thống dữ liệu (Data Systems Group)[11] của Đại học Waterloo, Canada, với những nghiên cứu chuyên sâu về dữ liệu lớn để xử lý, quản lý, phân tích và tìm kiếm trong xã hội thông tin hiện đại.

- Nhóm nghiên cứu hệ thống CSDL (Database Systems Group)[12] của trường Đại học kỹ thuật Munich, Đức, với những nghiên cứu về các phương thức mới quản trị CSDL, trong đó chú trọng khai thác mô hình CSDL trong bộ nhớ (in-memory database) và mô hình CSDL đồ thị để quản lý và khai thác dữ liệu quy mô lớn hiện nay.

[10] `https://sna.stanford.edu/`
[11] `https://uwaterloo.ca/data-systems-group/`
[12] `http://db.in.tum.de/`

- Nhóm nghiên cứu hệ thống CSDL (Database Systems)[13] của trường Đại học Wisconsin-Madison, Mỹ, với những nghiên cứu về các hệ quản trị CSDL mới, khoa học dữ liệu, trong đó có những nghiên cứu về ứng dụng lý thuyết đồ thị trong quản trị dữ liệu.

- Trung tâm nghiên cứu về toán dữ liệu lớn (Global Research Center for BigData Mathematics)[14] của Viện quốc gia tin học Nhật Bản (NII), với những nghiên cứu chuyên sâu về các mạng xã hội quy mô lớn để đề xuất những giải thuật phân tích đồ thị với tốc độ xử lý và có tính sáng tạo cao.

- Phòng thí nghiệm về giải thuật Web (Laboratory for Web Algorithms)[15] của trường Đại học Milano, Ý, với những nghiên cứu về xử lý và phân tích các đồ thị Web, mạng xã hội. Đây cũng là nơi tập hợp được nhiều nguồn dữ liệu liên quan đến các mạng xã hội và cung cấp công khai cho cộng đồng.

1.3 Mục tiêu, phạm vi nghiên cứu, đóng góp và bố cục của luận án

Với thực trạng đã đặt ra, trong luận án này chúng tôi quan tâm đến bài toán nghiên cứu đề xuất phương pháp tổ chức dữ liệu đồ thị phù hợp kết hợp cùng với những giải pháp song song hoá các phép toán trên đồ thị quy mô lớn cả về số cạnh lẫn số đỉnh để có thể tiến hành xử lý các truy vấn và phân tích trên đồ thị một cách hiệu quả nhất. Các phép toán được quan tâm trên đồ thị gồm những phép toán truy vấn khoảng cách ngắn nhất (với đồ thị động) và những phép tính độ đo trung tâm trong phân tích đồ thị (với đồ thị tĩnh).

1.3.1 Mục tiêu nghiên cứu

Từ bài toán đặt ra, mục tiêu chính của luận án là khảo sát, đánh giá các giải pháp hiện đại về xử lý các phép toán đồng thời trên đồ thị quy mô lớn; từ đó đề xuất phương pháp tổ chức dữ liệu đồ thị phù hợp và nâng cao hiệu năng thi hành các truy vấn đồng thời (cả về khoảng cách ngắn nhất và cập nhật) trên đồ thị động cũng như cải thiện hiệu năng tính một số độ đo trung tâm phục vụ phân tích đồ thị có quy mô lớn.

Mục tiêu này sẽ được thể hiện cụ thể thông qua các nội dung nghiên cứu chính trong luận án như sau:

1. Nghiên cứu, khảo sát và đánh giá một số phương pháp, kỹ thuật tổ chức dữ liệu đồ thị cũng như các phép toán cơ bản trên đồ thị.

[13] `https://database.cs.wisc.edu/`
[14] `https://bigdata.nii.ac.jp/wp/english/`
[15] `http://law.di.unimi.it/index.php`

2. Nghiên cứu xây dựng mô hình đặc tả bài toán xử lý các truy vấn khoảng cách ngắn nhất trên đồ thị động, quy mô lớn.

3. Đề xuất một số giải pháp để nâng cao hiệu năng thi hành các truy vấn khoảng cách ngắn nhất trên đồ thị động, quy mô lớn dựa trên cách tiếp cận tổ chức dữ liệu phù hợp và tính toán song song.

4. Nâng cao hiệu năng một số giải thuật tính các độ đo phục vụ các phép toán phân tích đồ thị dựa trên cách tiếp cận tổ chức dữ liệu phù hợp và tính toán song song.

5. Tiến hành cài đặt thử nghiệm các giải pháp đã xây dựng trong luận án; đánh giá và so sánh với một số giải pháp hiện có dựa trên những bộ dữ liệu chuẩn.

1.3.2 Phạm vi và phương pháp nghiên cứu

Về phạm vi, trong luận án này chúng tôi chỉ chú trọng đến bài toán nghiên cứu trên đồ thị không trọng số. Đối với bài toán xử lý các truy vấn khoảng cách ngắn nhất trên đồ thị động, chúng tôi quan tâm đến đồ thị có hướng, không trọng số với các phép toán thêm cạnh, xoá cạnh (từ đó có thể hình thành các phép toán thêm/xoá đỉnh) và truy vấn khoảng cách ngắn nhất giữa hai đỉnh. Đối với các phép toán hỗ trợ phân tích đồ thị quy mô lớn, chẳng hạn như các mạng xã hội, một số độ đo trung tâm sẽ được quan tâm, đề xuất giải pháp nâng cao hiệu năng tính toán các độ đo này trong luận án.

Với năng lực của hạ tầng tính toán tiếp cận được, hiện chúng tôi chưa thể tiến hành để giải quyết hiệu quả đối với đồ thị có quy mô quá lớn trên một tỷ đỉnh, chẳng hạn như dữ liệu mạng Facebook.

Về phương pháp nghiên cứu, trong luận án này chúng tôi sẽ kết hợp cả phương pháp nghiên cứu lý thuyết lẫn nghiên cứu thực nghiệm. Về nghiên cứu lý thuyết, chúng tôi sẽ tiến hành thu thập các tài liệu khoa học đã được công bố tại các nhà xuất bản, trường Đại học có uy tín trong và ngoài nước để từ đó phân tích, đánh giá những phương pháp, kỹ thuật cũng như kết quả thu được trong lĩnh vực liên quan đến bài toán nghiên cứu của luận án. Các đề xuất trong luận án cũng được chú trọng phân tích, đánh giá tường minh về tính đúng đắn, độ phức tạp về mặt lý thuyết. Về nghiên cứu thực nghiệm, chúng tôi sẽ áp dụng phương pháp thực nghiệm đối với toàn bộ các giải pháp, đề xuất của chúng tôi để kiểm nghiệm lại kết quả lý thuyết. Việc thực nghiệm cũng được tiến hành trên những bộ dữ liệu thường xuyên được cộng đồng nghiên cứu sử dụng và trên cùng nền tảng tính toán để có thể so sánh với những giải pháp khác đã được công bố.

1.4 Các đóng góp chính của luận án

Trong quá trình thực hiện luận án này, chúng tôi đã thu được những kết quả chính sau đây:

1. Mô hình hoá quá trình xử lý các truy vấn khoảng cách ngắn nhất trên đồ thị động, quy mô lớn dựa vào lịch thi hành phép toán đồng thời và dựa vào cấu trúc dữ liệu phù hợp cho phép nâng cao hiệu năng bộ nhớ đệm cache. Đóng góp này được công bố trong công trình được đăng trên kỷ yếu hội thảo quốc tế ICCCI năm 2017 [DPH2].

2. Đề xuất ba giải pháp (akGroup, akGroupPlus và bigGraph) để nâng cao hiệu năng thi hành các truy vấn đồng thời trên đồ thị động quy mô lớn với khả năng thi hành song song cả các truy vấn duyệt đồ thị lẫn cập nhật đồ thị. Cả ba giải pháp này đều dựa trên ý tưởng chính là (i) xây dựng cấu trúc dữ liệu đồ thị phù hợp để nâng cao hiệu năng của bộ nhớ đệm cache; (ii) lựa chọn hướng duyệt đồ thị một cách linh hoạt dựa không chỉ vào số lượng đỉnh con mà cả số lượng đỉnh cháu của mỗi hàng đợi; và (iii) đề xuất giải pháp song song hoá các truy vấn đồng thời, cả đối với các phép toán cập nhật lẫn truy vấn khoảng cách ngắn nhất trên đồ thị. Các kết quả này đã được chúng tôi công bố trong công trình [DPH1] tại hội thảo BDCAT về quản lý dữ liệu lớn năm 2016, hội thảo quốc tế ICCCI năm 2017 [DPH2] và công bố trong tạp chí quốc tế Transactions on Computational Collective Intelligence, Springer, năm 2018 [DPH3].

3. Xây dựng hai giải thuật nâng cao hiệu năng quá trình tính độ trung tâm gần và độ trung tâm trung gian trên đồ thị quy mô lớn với giải pháp bigGraph được xây dựng dựa trên việc (i) tổ chức dữ liệu đồ thị phù hợp và (ii) song song hoá các phép tính SSSP trên mỗi đỉnh của đồ thị. Kết quả này của chúng tôi đã được công bố trong kỷ yếu hội thảo quốc tế SoICT năm 2018 [DPH4].

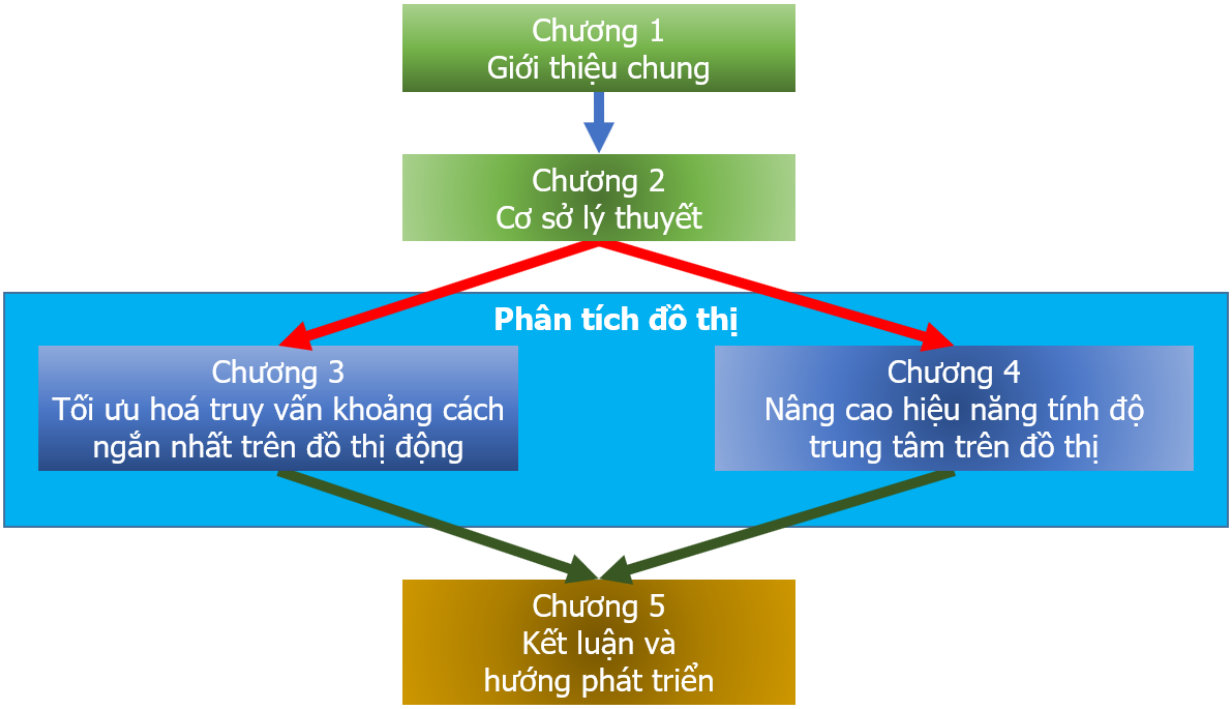
1.5 Tổ chức của luận án

Ngoài phần mở đầu giới thiệu chung, bố cục của luận án được tổ chức thành 5 chương được minh hoạ như hình 1.1. Nội dung chính của các chương như sau:

- Chương 1 giới thiệu chung về động lực nghiên cứu, mục tiêu và các nội dung chính của luận án. Ngoài ra, các nghiên cứu liên quan cũng như các đóng góp chính của luận án cũng được trình bày trong chương này.

- Chương 2 có nhiệm vụ trình bày các kiến thức cơ sở liên quan đến những nội dung nghiên cứu của luận án. Trong chương này, chúng tôi sẽ giới thiệu sơ lược về lý thuyết đồ thị, các phương pháp tổ chức dữ liệu đồ thị, các phép toán toán cơ bản trên đồ thị và một số những khái niệm cơ bản liên quan đến tính toán song song.

- Chương 3 của luận án giới thiệu về mô hình hoá các truy vấn tính khoảng cách ngắn nhất trên đồ thị động, quy mô lớn, từ đó chú trọng, đi sâu trình bày ba giải pháp chính được xây dựng trong luận án để nâng cao hiệu năng xử lý các truy vấn đồng thời tính khoảng cách ngắn nhất và cập nhật đồ thị quy mô lớn. Các giải pháp này được chứng



Hình 1.1: Lược đồ tổ chức luận án

minh tính đúng đắn, hiệu quả thông qua các thực nghiệm sử dụng những bộ dữ liệu được cộng đồng nghiên cứu thường sử dụng và đánh giá, so sánh với những công cụ tương tự.

- Chương 4 trình bày một số kết quả nghiên cứu về việc tính các độ đo trung tâm của đồ thị theo định hướng song song hoá kết hợp sử dụng cấu trúc dữ liệu đồ thị phù hợp. Hai giải pháp tính độ trung tâm gần và độ trung tâm trung gian đã được chúng tôi trình bày trong chương này. Các kết quả thực nghiệm minh chứng việc cải thiện hiệu năng của hai giải pháp đề xuất trong luận án thông qua đánh giá, so sánh với một số bộ công cụ tương tự cũng được tiến hành và trình bày cụ thể trong chương này.

- Chương 5 tóm lược lại các đóng góp chính của luận án và một số hướng phát triển trong tương lai.

Chương 2

CƠ SỞ LÝ THUYẾT

Trong chương này, luận án sẽ chú trọng trình bày những khái niệm lý thuyết cơ bản liên quan đến luận án, cụ thể là lý thuyết đồ thị, các phương pháp biểu diễn đồ thị cũng như các phép toán cơ bản trên đồ thị. Một số những khái niệm cơ bản liên quan đến tính toán song song cũng sẽ được trình bày trong luận án này.

2.1 Lý thuyết đồ thị

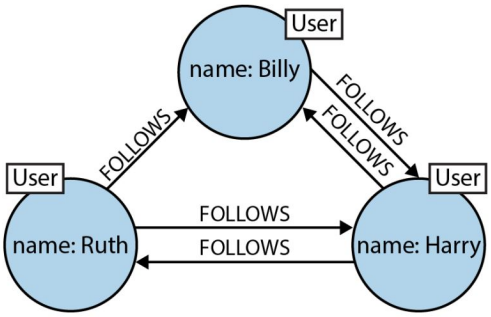
2.1.1 Khái niệm

Đồ thị là một cấu trúc dữ liệu linh hoạt, được thể hiện dưới dạng một tập các đỉnh (vertices) và các cạnh (edges), hay được gọi với thuật ngữ khác là tập các nút (nodes) và các quan hệ kết nối giữa chúng với nhau (relationships) [103][108]. Đồ thị cho phép biểu diễn các thực thể dưới dạng các đỉnh và các cách thức mà các thực thể đó liên quan đến nhau dưới dạng các mối quan hệ. Cấu trúc dễ hình tượng và đa mục đích này cho phép chúng ta có thể mô hình hóa tất cả các loại bài toán khác nhau, từ việc xây dựng tên lửa vũ trụ, đến một hệ thống đường, từ chuỗi cung cấp hoặc nguồn gốc thực phẩm, đến lịch sử y tế của người dân, và hơn thế nữa [48].

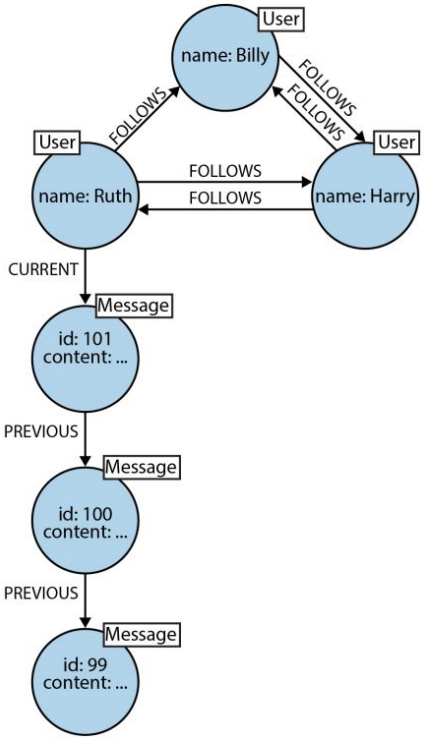
Định nghĩa 2.1. *Trong lý thuyết đồ thị, một đồ thị G, ký hiệu* $G = (V, E)$*, được cấu thành từ một tập các đỉnh V và một tập các cạnh E liên kết các đỉnh với* $E = \{(v_i, v_j)|v_i, v_j \in V\}$*. Số lượng phần tử các đỉnh* $|V|$ *và các cạnh* $|E|$ *được ký hiệu lần lượt là* n *và* m*.*

Đồ thị rất hữu ích trong việc phân tích và hiểu được sự đa dạng của các tập dữ liệu khoa học, dữ liệu của chính phủ, dữ liệu xã hội... Kỹ thuật quản lý dữ liệu theo mô hình quan hệ đã khiến chúng ta đi chệch ra khỏi miền ngôn ngữ tự nhiên, trước tiên bằng việc cố gắng biểu diễn thế giới thực dưới dạng một mô hình logic, rồi sau đó lại gắn nó vào một mô hình vật lý. Những phép biến đổi này chỉ ra sự bất hòa ngữ nghĩa giữa khái niệm của chúng ta về thế giới thực và thể hiện của cơ sở dữ liệu về thế giới đó. Với lý thuyết đồ thị, khoảng cách này co lại đáng kể khi cho phép thể hiện trực quan giữa mô hình dữ liệu và mô hình

vật lý trong thế giới thực. Chẳng hạn, dữ liệu của mạng xã hội Twitter dễ dàng được biểu thị dưới dạng đồ thị như minh hoạ ở Hình 2.1. Các quan hệ giữa các thành viên được thể hiện bởi mối quan hệ "theo dõi - follow" lẫn nhau. Các mối quan hệ này thể hiện bối cảnh ngữ nghĩa: cụ thể trong ví dụ này là Billy theo dõi Harry còn Harry theo dõi Billy và Ruth. Ruth và Harry cũng theo dõi nhau, nhưng Billy không thể hiện sự quan tâm đến các hoạt động của Ruth.



Hình 2.1: Minh họa về đồ thị xã hội

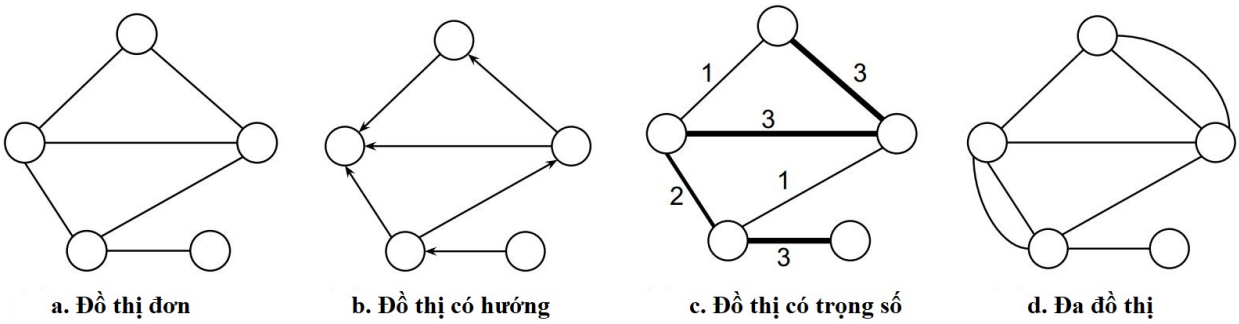


Hình 2.2: Minh họa việc xuất bản thông điệp

Tất nhiên, đồ thị thực sự của Twitter lớn hơn hàng trăm triệu lần so với ví dụ trong hình 2.1, nhưng nó hoạt động trên cùng một nguyên tắc. Hình 2.2 minh hoạ các thông điệp do Ruth xuất bản. Rõ ràng các tương tác được thể hiện hiệu quả trong mô hình đồ thị: Ruth đã xuất bản một chuỗi thông điệp; thông điệp gần đây nhất có thể được tìm thấy nhờ nhãn CURRENT còn nhãn PREVIOUS sau đó tạo ra một dòng thời gian của các bài viết.

2.1.2 Kiểu đồ thị

Một số các kiểu đồ thị được thể hiện trong Hình 2.3 dưới đây [108]:



Hình 2.3: Một số kiểu đồ thị cơ bản

Định nghĩa 2.2. ***Đồ thị vô hướng (undirected graph)****: là đồ thị trong đó E là tập các cặp không thứ tự chứa các đỉnh phân biệt. Hai đỉnh thuộc một cạnh được gọi là các đỉnh đầu cuối của cạnh đó. Như vậy, trong đồ thị vô hướng, nếu $(v_i, v_j) \in E$ thì $(v_j, v_i) \in E$ và $(v_j, v_i) \equiv (v_i, v_j)$.*
Đồ thị có hướng (directed graph/digraph)*: là đồ thị trong đó E là tập các cặp có thứ tự chứa các đỉnh, được gọi là các cạnh có hướng. Cạnh $(v_i, v_j) \in E$ được coi là có hướng từ v_i tới v_j; v_i được gọi là điểm đầu/gốc và v_j được gọi là điểm cuối/ngọn của cạnh.*

Định nghĩa 2.3. ***Đồ thị có trọng số (weighted graph)****: là đồ thị mà mỗi cạnh đều được gán một trọng số. Ngược lại, khi không xét trọng số cho các cạnh, chúng ta có* ***đồ thị không trọng số (unweighted graph)****.*

Định nghĩa 2.4. ***Đa đồ thị (multi-graph)****: là đồ thị có nhiều cạnh giữa hai đỉnh, kể cả có thứ tự lẫn không có thứ tự.*
Đơn đồ thị (single graph)*: là đồ thị vô hướng, không có trọng số và không có khuyên lẫn cạnh song song.*

Định nghĩa 2.5. ***Đồ thị tượng trưng (attributed graph)****: là đồ thị mà cạnh và đỉnh có thể chứa cặp giá trị gắn kèm để mô tả thông tin kèm theo với các thực thể. Đồ thị có trọng số cũng có thể xem như một trường hợp của đồ thị tượng trưng.*
Đồ thị nhãn (labeled graph)*: là trường hợp đặc biệt của đồ thị tượng trưng mà mỗi nhãn có thể mô tả đỉnh và/hoặc cạnh để biểu thị kiểu hoặc loại. Chẳng hạn, một mạng cộng tác có thể có các tác giả, địa điểm và các ấn phẩm như các loại đỉnh, trong khi trích dẫn, đồng tác giả, kiểu xuất bản có thể là một số loại cạnh. Các loại đỉnh và cạnh khác nhau như vậy được xác định bằng nhãn.*
Đồ thị thuộc tính (property graph)*: là đồ thị kết hợp các đặc trưng của đa đồ thị nhãn có hướng, tượng trưng.*

Định nghĩa 2.6. ***Đồ thị hai phía (bipartite graph)****: là đồ thị mà tập đỉnh V có thể chia thành hai tập con không giao nhau V_1 và V_2 đảm bảo mỗi cạnh của E có một đỉnh trong V_1 và một đỉnh trong V_2. Trong đồ thị hai phía, không thể có các cạnh nối các đỉnh trong cùng một tập con.*

Định nghĩa 2.7. ***Đồ thị liên thông (connected graph)****: là đồ thị nếu luôn tồn tại đường đi đối với mọi cặp đỉnh $u, v \in V$. Ngược lại, đồ thị đó sẽ được gọi là* ***đồ thị không liên thông (disconnected graph)****.*

Đồ thị đầy đủ (complete graph)*: là đồ thị vô hướng mà mỗi cặp đỉnh được kết nối bởi một cạnh. Như vậy, trong đồ thị này, số cạnh $|E|$ được xác định bằng công thức $|E| = |V| * (|V| - 1)/2$ với $|V|$ là số đỉnh.*

Đồ thị có hướng đầy đủ (complete digraph)*: là đồ thị có hướng mà mỗi cặp đỉnh được kết nối bởi một cặp cạnh tương ứng với mỗi hướng. Trong đồ thị có hướng đầy đủ, số cạnh được xác định bằng công thức $|E| = |V| * (|V| - 1)$.*

Định nghĩa 2.8. ***Đồ thị con (subgraph)****: đồ thị $G_0 = (V_0, E_0)$ là đồ thị con của $G = (V, E)$ (ký hiệu $G_0 \subseteq G$) nếu như $V_0 \subseteq V$ và $E_0 \subseteq E$.*

2.1.3 Các đặc điểm chính của đồ thị

Ngoài các khái niệm liên quan đến các kiểu đồ thị trên, một số các đặc tính sau hay được sử dụng trong phân tích đồ thị.

Định nghĩa 2.9. ***Tập đỉnh láng giềng (neighbourhoods)****: của một đỉnh v là tập N các đỉnh liền kề (có cạnh nối) với v. Trong đồ thị có hướng, tập đỉnh láng giềng đến (incoming nodes - N_{in}) sẽ khác biệt với tập đỉnh láng giềng đi (outgoing nodes - N_{out})*

Định nghĩa 2.10. ***Bậc của một đỉnh (degree)****: là số cạnh liên kết của một đỉnh, ký hiệu là $d(v)$. Trong đồ thị có hướng, các cạnh đi và cạnh đến của một đỉnh sẽ cho phép xác lập bậc đi $d_{out}(v)$ (outgoing degree) và bậc đến $d_{in}(v)$ (incoming degree) của v.*

Định nghĩa 2.11. ***Đường kính đồ thị (graph diameter)****: là giá trị tương ứng với độ dài đường đi ngắn nhất lớn nhất giữa hai đỉnh bất kỳ trong đồ thị :*

$$d = \max_{u,v \in V} ShortestDistance(u, v)$$

Định nghĩa 2.12. ***Thành phần liên thông (connected component)****: là đồ thị con lớn nhất của một đồ thị vô hướng mà bất kì cặp đỉnh nào trong đồ thị con đó đều có đường đi đến nhau. (từ đó, đồ thị liên thông luôn có đúng một thành phần liên thông chính là toàn bộ đồ thị).*

Định nghĩa 2.13. ***Thành phần liên thông mạnh/yếu (strongly/weakly connected component)****: đối với đồ thị có hướng, thành phần liên thông mạnh là đồ thị con lớn nhất*

mà luôn tồn tại đường đi có hướng đối với mọi cặp đỉnh trong đồ thị con đó. Trong trường hợp chỉ tồn tại đường đi vô hướng (tức không xét đến hướng của các cạnh) giữa các cặp đỉnh, đồ thị con lớn nhất đó được gọi là thành phần liên thông yếu.

2.2 Biểu diễn đồ thị

Với khả năng mô hình hoá một cách trực quan và hiệu quả cả các thực thể lẫn các mối liên hệ giữa chúng, lý thuyết đồ thị được ứng dụng vào rất nhiều lĩnh vực khác nhau. Một số các lĩnh vực điển hình hiện nay sử dụng đồ thị để mô hình hoá dữ liệu có thể kể đến như mạng xã hội, mạng sinh học, mạng phân phối nội dung, mạng lưới giao thông, mạng ngữ nghĩa... [96]. Trong mỗi lĩnh vực cụ thể, để có thể phục vụ các phép toán xử lý và phân tích chuyên biệt, cần phải có những phương pháp, kỹ thuật tổ chức dữ liệu đồ thị cho phù hợp. Việc tổ chức dữ liệu đồ thị phải đảm bảo cho các thao tác cơ bản, chẳng hạn như tìm các đỉnh lân cận của một đỉnh, kiểm tra hai đỉnh xem có liên kết hay không, cập nhật thêm/xoá đỉnh/cạnh... Trong phần này, chúng ta sẽ làm rõ một số phương pháp đang được sử dụng hiện nay để biểu diễn dữ liệu đồ thị.

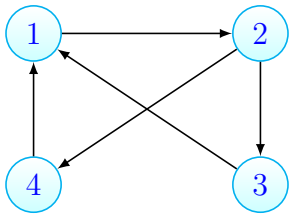
2.2.1 Danh sách các cạnh

Phương pháp đơn giản nhất để biểu diễn dữ liệu của một đồ thị là sử dụng một danh sách các cạnh. Danh sách này phải chứa toàn bộ các cạnh u, v của đồ thị G.

Với phương pháp này, việc tổ chức dữ liệu của đồ thị được cài đặt rất đơn giản, không gian bộ nhớ cần để lưu trữ đồ thị tương đối bé. Tuy nhiên, việc thực hiện các phép toán cơ bản sẽ có độ phức tạp rất cao, chẳng hạn tìm các đỉnh liền kề của một đỉnh vẫn cần phải duyệt toàn bộ danh sách này.

2.2.2 Ma trận liền kề

Dữ liệu đồ thị có thể được tổ chức dưới dạng một ma trận liền kề (adjecency matrix) A gồm n hàng và n cột, với $n = |V|$. Chẳng hạn, đối với đồ thị đơn giản có hướng, không trọng số G, có thể tổ chức A đảm bảo $A(i, j) = 1$ nếu $(i, j) \in E$ và 0 nếu ngược lại. Với đồ thị có trọng số, $A(i, j)$ sẽ được gắn luôn giá trị trọng số w của cạnh (i, j), tức $A(i, j) = w$. Trong đồ thị vô hướng, ma trận A sẽ là ma trận đối xứng qua đường chéo. Hình 2.4 dưới đây minh hoạ ma trận liền kề đối với đồ thị tương ứng.



	1	*2*	*3*	*4*
1	0	1	0	0
2	0	0	1	1
3	1	0	0	0
4	1	0	0	0

Hình 2.4: Đồ thị có hướng và ma trận liền kề

Đối với phương pháp biểu diễn đồ thị bằng ma trận liền kề, ma trận được gọi là thưa (sparse) nếu phần lớn các phần tử của nó bằng 0. Phần lớn các đồ thị có quy mô số đỉnh lớn thường mà sử dụng phương pháp này đều là ma trận liền kề thưa, chẳng hạn như các mạng xã hội.

Ma trận liền kề cho phép thi hành các phép toán cơ bản như thêm/xoá cạnh rất hiệu quả: độ phức tạp của các phép toán đó là $O(1)$. Tuy nhiên, thêm/xoá đỉnh lại cần phải tổ chức lại đồ thị. Ưu điểm của phương pháp biểu diễn này còn nằm ở điểm nhiều phép toán trên đồ thị hoàn toàn có thể quy về một loạt các hàm thao tác trên ma trận. Hạn chế của phương pháp nằm ở độ phức tạp về không gian là $O(n^2)$, ngay cả với các ma trận thưa và không phụ thuộc vào số cạnh. Điều này dẫn đến với mạng xã hội chẳng hạn, khi số đỉnh lên đến hàng tỷ thì phương pháp biểu diễn này hoàn toàn không khả thi.

2.2.3 Danh sách liền kề

Trong phương pháp biểu diễn đồ thị này, thông tin các cạnh của mỗi đỉnh trong đồ thị được tổ chức bằng một danh sách các đỉnh liền kề với nó. Như vậy, với đồ thị vô hướng, một cạnh thể hiện quan hệ giữa đỉnh u với v thì danh sách liền kề của u sẽ có v và ngược lại. Hình 2.5 dưới đây minh hoạ biểu diễn dạng danh sách liền kề đối với đồ thị được minh hoạ trong hình 2.4.

1	⇒	2		
2	⇒	3	⇒	4
3	⇒	1		
4	⇒	1		

Hình 2.5: Danh sách liền kề

Với cách tổ chức dữ liệu này, việc xác định các đỉnh liền kề của đỉnh v chỉ đơn thuần là duyệt danh sách liền kề đỉnh v. Các phép toán thêm/xoá đỉnh cũng rất đơn giản: bổ sung thêm đỉnh tương ứng với việc tạo một danh sách mới trống cho đỉnh đó; xoá một đỉnh v thì

ngoài loại danh sách tương ứng đỉnh đó cần duyệt tất cả các đỉnh trong danh sách đó để loại v khỏi tất cả các danh sách có v là liền kề.

Phương pháp tổ chức theo danh sách liền kề có độ phức tạp không gian đúng bằng số đỉnh và cạnh $O(|V| + |E|)$, và rõ ràng phương pháp này rất hiệu quả để biểu diễn đồ thị thưa so với phương pháp ma trận liền kề.

2.2.4 Ma trận liên thuộc

Ma trận liên thuộc (Incidence Matrix) là một biến thể của phương pháp biểu diễn ma trận liền kề. Trong phương pháp này, đồ thị G sẽ được thể hiện bằng ma trận I với $n = |V|$ hàng, và $m = |E|$ cột. Giá trị $I(i, j) = 1$ trong ma trận thể hiện cạnh j có đỉnh i, và ngược lại thì $I(i, j) = 0$. Với đồ thị có hướng, $I(i, j) = 1$ thể hiện i là đỉnh đến (outgoing) của cạnh j và $I(i, j) = -1$ thì i sẽ là đỉnh đến (incoming). Hình dưới đây thể hiện ma trận liên thuộc của đồ thị minh hoạ trong hình 2.4 với $e1 = (1, 2); e2 = (2, 3); e3 = (2, 4); e4 = (3, 1)$ và $e5 = (4, 1)$:

	e1	*e2*	*e3*	*e4*	*e5*
1	1	0	0	-1	-1
2	-1	1	1	0	0
3	0	-1	0	1	0
4	0	0	-1	0	1

Hình 2.6: Ma trận liên thuộc biểu diễn đồ thị

Với phương pháp này, độ phức tạp không gian của ma trận liên thuộc là $O(|V| * |E|)$. Các phép toán cơ bản trên đồ thị nhìn chung có độ phức tạp tính toán tương đương ma trận liền kề.

2.2.5 Ma trận hàng thưa nén

Ma trận hàng thưa nén (Compressed Sparse Row -CSR) là một biến thể của ma trận liền kề với mục đích truy cập nhanh các hàng. Phương pháp này được đưa ra từ những năm 1967 [17], theo đó đồ thị G sẽ được biểu diễn bởi ma trận A thông qua ba mảng một chiều tương ứng chứa các giá trị khác không, giá trị hàng và chỉ mục cột. Đặt NNZ (number of non-zero) là số các giá trị khác không trong A, ma trận $n \times n$ (với $n = |V|$) sẽ được tổ chức dưới dạng hàng thông qua ba mảng (AN, IA, JA) như sau:

- Mảng AN có độ dài NNZ lưu các giá trị khác 0 của A theo thứ tự hàng (row-major order), từ tính từ trái qua phải, từ trên xuống dưới.

- Mảng IA có $n+1$ phần tử thể hiện giá trị cộng dồn các phần tử khác 0 trong mỗi hàng. Cụ thể được tính bằng công thức $IA[0] = 0; IA[i] = IA[i-1] +$ số_phần_tử_khác_0_tại_hàng_i : $i \in (1..n)$
- Mảng JA cũng có NNZ phần tử để lưu chỉ mục cột của các phần tử khác 0 trong A

Chẳng hạn, với đồ thị được minh hoạ bằng ma trận ở hình 2.4, ba mảng CSR với $NNZ = 5; n = 5$ sẽ được thể hiện như sau:

$$\begin{aligned} AN &= [1\ 1\ 1\ 1\ 1] \\ IA &= [0\ 1\ 3\ 4\ 5] \\ JA &= [2\ 3\ 4\ 1\ 1] \end{aligned}$$

Hình 2.7: Ma trận hàng thưa nén

Như vậy, với cách biểu diễn CSR như trên, ta chỉ cần lưu 15 giá trị trong 3 mảng thay vì 16 giá trị như trong ma trận liền kề. Thực tế, khi xét với ma trận kích thước $n \times m$, CSR biểu diễn hiệu quả hơn khi có số $NNZ < (n * (m-1) - 1)/2$. Phương pháp biểu diễn này nhìn chung cho phép thi hành nhanh các phép toán nhân vector ma trận [95].

Trong thực tế, còn có thêm các phương pháp tổ chức dữ liệu đồ thị không hoàn toàn đồng nhất với các phương pháp đã nêu. Tuỳ thuộc vào các yêu cầu thực tế khác nhau liên quan đến tối ưu về không gian lưu trữ, tối ưu cho các phép toán thêm/xoá/cập nhật hay tối ưu cho xử lý truy vấn mà có những biến thể khác nhau của các truy vấn trên [102]. Ngoài ra, trong một số trường hợp đặc biệt, các cách thức biểu diễn dữ liệu đồ thị nêu trên không phù hợp, chẳng hạn khi biểu diễn đa đồ thị hay biểu diễn đồ thị có các thuộc tính gắn với các đỉnh/cạnh. Khi đó, chúng ta lại cần phải có những cách thức biểu diễn bổ sung đối với các đồ thị đó.

2.3 Các phép toán chính trên đồ thị

Trong phần này, chúng tôi sẽ trình bày những kết quả tóm lược các phép toán chính trên đồ thị. Các phép toán này được chia làm hai loại chính: (i) duyệt đồ thị, và (ii) phân tích đồ thị [18].

Với lý thuyết đồ thị, việc tiến hành các truy vấn trên đồ thị bao giờ cũng phải dựa trên quá trình thăm và duyệt đồ thị. Điều này xuất phát từ bản chất biểu diễn dữ liệu của đồ thị: các đỉnh thường được sử dụng để đại diện thực thể trong khi các cạnh thường được đặc tả các quan hệ giữa chúng. Chính vì thế, trong CSDL đồ thị cũng như các bài toán được mô hình hoá bằng đồ thị, các phép toán duyệt đồ thị có thể được xem như là trái tim của các thành phần xử lý truy vấn.

Ngoài việc duyệt đồ thị, trong quá trình phân tích đồ thị, một số các phép toán cũng được sử dụng chẳng hạn như tính khoảng cách giữa hai đỉnh, tìm đường đi ngắn nhất, tính

độ trung tâm, phân cụm đồ thị... Trong chương này chúng tôi cũng sẽ trình bày các định nghĩa cụ thể của các phép toán phân tích đồ thị này.

2.3.1 Duyệt đồ thị

Định nghĩa 2.14. *Duyệt đồ thị (hay đôi khi còn gọi tìm kiếm trên đồ thị) là quá trình thăm (kiểm tra hoặc/và cập nhật) các cạnh và các đỉnh trong đồ thị. Quá trình duyệt đồ thị thường được phân loại dựa theo thứ tự các đỉnh được khám phá.*

Khi duyệt đồ thị, vết các đỉnh và cạnh đã thăm hình thành các khái niệm "walk-lối đi", "path-đường đi", "cycle-chu trình" và được định nghĩa như sau [55]:

Định nghĩa 2.15. *Lối đi (walk) là chuỗi liên tiếp các đỉnh và cạnh của đồ thị. Khi duyệt đồ thị từ đỉnh u đến đỉnh v, các đỉnh và cạnh đã thăm trong quá trình duyệt hình thành lối đi giữa u và v.*

Định nghĩa 2.16. *Đường đi (path) là lối đi trong đồ thị mà không có đỉnh hay cạnh nào được lặp lại.*

Định nghĩa 2.17. *Chu trình (cycle) là đường đi mà đỉnh đầu và cuối trùng nhau.*

Trong quá trình duyệt đồ thị, có thể có những đỉnh được thăm nhiều hơn một lần nếu như đỉnh đó có nhiều đỉnh liền kề với nó. Do đó, nếu đồ thị càng "dầy" (dense), việc duyệt lại lại càng phổ biến hơn; ngược lại, với đồ thị thưa, thời gian duyệt lại một đỉnh sẽ ít đi. Chính vì điều này mà các phương pháp duyệt đồ thị bao giờ cũng phải lưu lại danh sách các đỉnh đã duyệt và trạng thái đã duyệt hết các đỉnh liền kề để có thể khám phá hết được các đỉnh trong đồ thị.

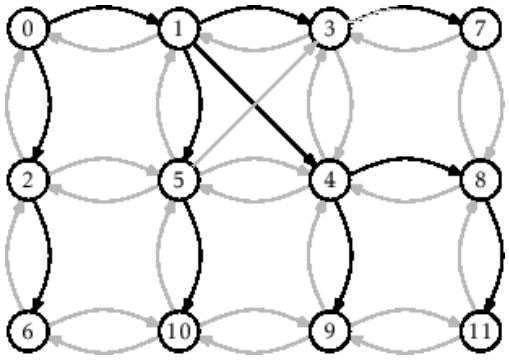
Duyệt đồ thị thường được phân thành hai phương pháp chính: duyệt theo chiều rộng trước và duyệt theo chiều sâu trước. Với giả thiết đồ thị G được biểu diễn theo phương pháp sử dụng danh sách liền kề, chúng ta sẽ tìm hiểu kỹ hơn hai phương pháp duyệt này dưới đây:

2.3.1.1 Duyệt theo chiều rộng trước - BFS

Định nghĩa 2.18. *Duyệt theo chiều rộng trước (Breadth-First Search - BFS) là giải thuật để duyệt cấu trúc dữ liệu đồ thị. Giải thuật này cho phép từ một đỉnh trong đồ thị, gọi là đỉnh gốc, khám phá tất cả các đỉnh liền kề có cùng độ sâu so với đỉnh gốc trước khi chuyển đến các đỉnh ở độ sâu tiếp theo [33].*

Giải thuật này lần đầu được giới thiệu trong công bố của Edward F. Moore [76] năm 1959 để tìm đường đi ngắn nhất qua mê cung, và được phát triển sau đó bởi C. Y. Lee để xây dựng giải thuật định tuyến mạch điện năm 1961 [57]. Với cách tiếp cận duyệt các đỉnh liền kề như thế, BFS có thể được sử dụng để tính khoảng cách ngắn nhất (shortest distance)

giữa hai đỉnh bất kỳ trong đồ thị. Ngoài ra, BFS cũng được sử dụng để giải bài toán tìm các đường đi từ một đỉnh nguồn đến tất cả các đỉnh còn lại của đồ thị. Hình sau minh hoạ quá trình duyệt theo BFS đối với một đồ thị có hướng với đỉnh gốc là 0 [72]:



Hình 2.8: Ví dụ về duyệt theo chiều rộng trước

Xét bài toán tìm đỉnh t từ đỉnh gốc v trong đồ thị G thỏa mãn điều kiện tìm kiếm C (chẳng hạn, khoảng cách từ v đến t là 10). Mã giả cho giải thuật duyệt đồ thị G từ đỉnh gốc v theo chiều rộng trước được minh hoạ như sau [83]:

Thuật toán 2.1: $BFS(G, v)$: Mã giả phương pháp duyệt theo chiều rộng trước

```
1  Function BFS(G, v)
       Input: G = (V, E); v ∈ V; điều kiện tìm kiếm C
       Output: t ∈ V thoả mãn điều kiện tìm kiếm C
2      if v thoả mãn điều kiện tìm kiếm then return v ;
3      Tạo mảng lưu vết đỉnh đã duyệt Seen[|V|] và khởi tạo giá trị False ;
4      Seen[v] = True ;                          /* đánh dấu v đã được duyệt */
5      Tạo hàng đợi Q và đưa v vào Q ← v ;
6      while Q ≠ ∅ do
7          t ← Q.dequeue() ;
8          forall u ∈ G.adjacentNodes(t) do      /* xét các đỉnh liền kề của t */
9              if !Seen[u] then
10                 if u thoả mãn điều kiện tìm kiếm C then return u ;
11                 Seen[u] = True ;                    /* đánh dấu u đã duyệt */
12                 Q ← u ;                              /* đưa u vào hàng đợi */
13             end
14         end
15     end
16     return null ;
```

Trong giải thuật 2.1, chúng ta sử dụng một hàng đợi Q để lưu những đỉnh liền kề với đỉnh đang thăm hiện thời mà chưa được duyệt. Mỗi khi thăm một đỉnh, chúng ta sẽ đánh dấu lại đỉnh đã duyệt bằng cách sử dụng một vector kiểu bool có độ lớn bằng số đỉnh của

G, nhằm tránh việc duyệt lại hay thậm chí tạo vòng lặp bất tận. Với phương pháp tổ chức dữ liệu đồ thị dưới dạng danh sách đỉnh liền kề, mỗi khi duyệt một đỉnh, chúng ta sẽ đưa toàn bộ các đỉnh liền kề với nó (xác định dựa vào hàm $G.adjacentNodes(v)$) còn chưa được thăm vào trong hàng đợi Q. Như vậy, với việc tổ chức hàng đợi các đỉnh sẽ duyệt, BFS sẽ cho phép chúng ta thăm các đỉnh tuần tự theo độ sâu kể từ đỉnh gốc cần tìm.

Định lý 2.1. *Độ phức tạp tính toán của phương pháp BFS duyệt theo chiều rộng trước là* $O(|V| + |E|)$.

Thật vậy, với giải thuật 2.1, số đỉnh tối đa chúng ta có thể đưa vào hàng đợi Q là V. Ngoài ra, với vòng lặp $ForAll$, số cạnh tối đa chúng ta thăm trong giải thuật này là V. Do đó, độ phức tạp tính toán theo thời gian của giải thuật này là $O(|V| + |E|)$.

Phương pháp duyệt BFS cho phép giải quyết được rất nhiều bài toán đồ thị, chẳng hạn như: kiểm tra khả năng hai phía (bipartiteness) của đồ thị; tìm đường đi ngắn nhất về số cạnh giữa hai đỉnh; tính luồng cực đại trong mạng theo phương pháp Ford-Fulkerson; xây dựng hàm lõi theo giải thuật Aho-Corasick; ... [38].

2.3.1.2 Duyệt theo chiều sâu trước - DFS

Định nghĩa 2.19. *Duyệt theo chiều sâu trước (Depth-First Search - DFS) là giải thuật để duyệt cấu trúc dữ liệu đồ thị, cho phép từ một đỉnh trong đồ thị, gọi là đỉnh gốc, khám phá càng xa càng tốt dọc theo từng đỉnh liền kề trước khi quay trở lại. Hay khác với BFS khám phá các đỉnh anh em cùng mức trước, DFS cho phép khám phá các đỉnh con trước khi quay lại khám phá các đỉnh anh em [33].*

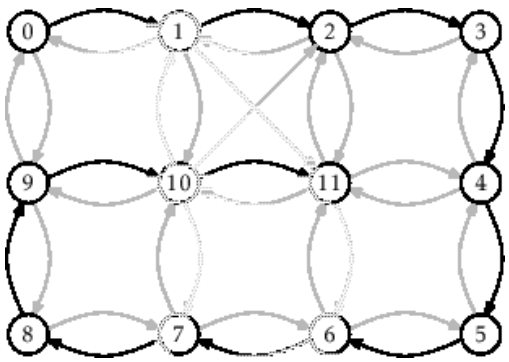
Với cách tiếp cận duyệt đồ thị như vậy, giải thuật duyệt DFS thường sử dụng ngăn xếp (stack) để lưu các đỉnh chưa duyệt. Xét bài toán như ở mục trên: tìm đỉnh t từ đỉnh gốc v trong đồ thị G thỏa mãn điều kiện tìm kiếm C. Khi đó, mã giả của DFS được minh hoạ như

sau:

Thuật toán 2.2: $DFS(G, v)$: Mã giả phương pháp duyệt theo chiều sâu trước

```
1  Function DFS(G,v)
       Input: G = (V, E); v ∈ V; điều kiện tìm kiếm C
       Output: u ∈ V thoả mãn điều kiện tìm kiếm C
2      if v thoả mãn điều kiện tìm kiếm then return v;
3      Tạo mảng lưu vết đỉnh đã duyệt Seen[|V|] và khởi tạo giá trị False ;
4      Seen[v] = True ;                          /* đánh dấu v đã được duyệt */
5      Tạo ngăn xếp S và đưa v vào S ← v ;
6      while S ≠ ∅ do
7          t ← S.pop() ;
8          forall u ∈ G.adjacentNodes(v) do      /* xét các đỉnh liền kề của t */
9              if !Seen[u] then
10                 if u thoả mãn điều kiện tìm kiếm C then return u ;
11                 S ← u ;                              /* đưa u vào ngăn xếp */
12                 Seen[u] = True ;                  /* đánh dấu u đã được duyệt */
13             end
14         end
15     end
16     return null ;
```

Với DFS, ta sẽ bắt đầu thăm đỉnh "gốc" v đã chọn. Nếu đỉnh này chưa thoả mãn điều kiện tìm kiếm, chúng ta sẽ chọn lần lượt các đỉnh liền kề u với v để duyệt tiếp. Với mỗi đỉnh u chưa được thăm (dựa vào vector kiểu bool có độ lớn bằng số đỉnh của V), chúng ta sẽ duyệt đệ quy để thăm tiếp đỉnh con của u. Sau khi thăm xong tất cả các đỉnh con của u mà không tìm thấy kết quả mong muốn, DFS mới quay lui lại để xét tiếp các đỉnh cùng mức (hay các đỉnh anh em) với u. Chính vì vậy, DFS sẽ có phép duyệt các đỉnh trong đồ thị theo chiều sâu trước. Hình dưới đây minh hoạ quá trình duyệt các đỉnh theo DFS từ đỉnh gốc 0 [72]:



Hình 2.9: Ví dụ về duyệt theo chiều sâu trước

Định lý 2.2. *Độ phức tạp tính toán của phương pháp DFS duyệt theo chiều sâu trước cũng là* $O(|V| + |E|)$.

Thật vậy, cũng tương tự như phương pháp duyệt BFS, số đỉnh tối đa chúng ta có thể đưa vào hàng đợi Q trong giải thuật 2.2 là V. Ngoài ra, với vòng lặp $ForAll$, số cạnh tối đa chúng ta thăm trong giải thuật này là V. Do đó, độ phức tạp tính toán theo thời gian của giải thuật này là $O(|V| + |E|)$.

Giải thuật DFS cũng cho phép giải quyết hiệu quả nhiều lớp bài toán khác nhau dựa trên lý thuyết đồ thị, chẳng hạn: tìm các thành phần liên thông trong đồ thị; tìm các cầu nối trong đồ thị; phát hiện các chu trình trong đồ thị; sinh các từ để xác định tập giới hạn nhóm ... [33].

2.3.2 Phân tích đồ thị

Lý thuyết đồ thị được ứng dụng trong nhiều lĩnh vực khác nhau hiện nay. Tuỳ thuộc vào mỗi lĩnh vực ứng dụng đó mà việc phân tích đồ thị được tiến hành với những phương pháp và kỹ thuật khác nhau. Một số miền lĩnh vực có thể ứng dụng lý thuyết đồ thị để phân tích có thể liệt kê như dưới đây:

- Mạng xã hội (social network): đồ thị được ứng dụng để mô hình hoá mạng xã hội với các thực thể xã hội và các quan hệ giữa chúng [26]. Các thực thể này thường là con người, tuy nhiên có thể là nhóm, tổ chức, quốc gia, Websites hay các ấn bản khoa học...

- Mạng sinh học (biological network): là đồ thị được dùng để mô hình hoá hệ thống sinh học với các thành phần như loài (species), tế bào, phân tử, gen, tín hiệu, chuyển hoá... Mạng sinh học cho phép biểu diễn toán học về các kết nối trong nghiên cứu sinh thái, tiến hoá và sinh lý học [87]

- Mạng liên kết (link network): là đồ thị dùng để thể hiện các quan hệ giữa các đối tượng nói chung. Từ đó, dựa trên phân tích liên kết, các ngân hàng, cơ quan bảo hiểm có thể phát hiện được gian lận; các nhà cung cấp mạng viễn thông có thể quy hoạch mạng phù hợp hơn; ngành y học và dịch tễ học có thể kiểm soát được dịch bệnh...[77]

Một số các khái niệm, phương pháp hay được sử dụng trong phân tích đồ thị sẽ được trình bày cụ thể hơn trong các phần sau.

2.3.2.1 Tính khoảng cách

Trong đồ thị G, việc xác định đường đi ngắn nhất từ một đỉnh u đến đỉnh v sẽ cho phép tính được khoảng cách giữa chúng. Tuỳ thuộc vào mỗi kiểu đồ thị mà cách tính khoảng cách giữa u và v được xây dựng khác nhau.

- *Khoảng cách giữa hai đỉnh u và v, ký hiệu $dis(u, v)$ của đồ thị không trọng số G là số cạnh trong đường đi ngắn nhất, tối ưu nhất từ u đến v.*
- *Với đồ thị G có trọng số, khoảng cách từ đỉnh u đến v, kí hiệu $dis(u, v)$, là tổng tất cả các giá trị trọng số trong đường đi ngắn nhất, tối ưu nhất từ u đến v.*
- *Khoảng cách Euclidean giữa hai đỉnh u và v là số đỉnh liền kề (láng giềng) chung của cả u và v.*

Đối với cả hai loại đồ thị nêu trên, nếu không tồn tại đường đi ngắn nhất, khoảng cách từ u đến u được coi như không xác định. Trong trường hợp $u \equiv v$ thì $dis(u, v) = 0$. Thực tế, có thể có nhiều đường đi ngắn nhất từ u đến v, tuy nhiên, $dis(u, v)$ luôn có giá trị không đổi.

Ngoài các khái niệm khoảng cách nêu trên, trong phân tích đồ thị đôi khi còn sử dụng độ đo khoảng cách Euclidean. Khoảng cách này thông thường được dùng để xác định độ tương đồng giữa hai đỉnh. Trong các mạng xã hội, khoảng cách Euclidean giữa hai người được xác định chính là số bạn chung giữa hai người đó.

Từ khái niệm khoảng cách nêu trên, trong phân tích đồ thị, chúng ta sẽ tính được đường kính đồ thị theo đúng định nghĩa đã trình bày ở mục trên.

2.3.2.2 Đường đi ngắn nhất

Trong lý thuyết đồ thị, đường đi ngắn nhất từ đỉnh u đến v được định nghĩa là đường đi có khoảng cách $dis(u, v)$ ngắn nhất. Với đồ thị không có trọng số, đường đi ngắn nhất chính là đường đi có số cạnh nhỏ nhất từ u đến v.

Trong thực tế, chúng ta gặp rất nhiều bài toán ứng dụng liên quan đến tìm đường đi ngắn nhất. Chẳng hạn như tìm đường trong hệ thống giao thông; tìm số người ít nhất để hình thành quan hệ giữa hai người trong mạng xã hội; tìm đường đi có độ trễ nhỏ nhất (min-delay path problem); ... [2]

Để tìm đường đi ngắn nhất, Dijkstra là giải thuật hiệu quả nhất đối với trường hợp đồ thị có trọng số không âm [28].Trong khi đó, với đồ thị không trọng số, giải thuật **BFS** được coi như giải thuật hiệu quả nhất để xác định đường đi cũng như khoảng cách ngắn nhất từ u đến v [53]. Một biến thể của BFS cho phép cải thiện được hiệu năng tính khoảng cách ngắn nhất là thực hiện tính BFS từ hai cả hai đỉnh nguồn và đích. Cách tiếp cận này hình thành giải thuật duyệt theo chiều rộng cả hai hướng bBFS (bi-directional BFS) [47] với mã

giả được minh hoạ như sau:

Thuật toán 2.3: Giải thuật tính khoảng cách ngắn nhất sử dụng bBFS

```
1  Function BBFSDistance(G,u,v)
       Input: G = (V, E); u, v ∈ V
       Output: khoảng cách ngắn nhất giữa hai đỉnh u và v
2      if u == v then return 0 ;
3      Tạo mảng đánh dấu các đỉnh đã duyệt từ nguồn Seen_out[|V|], từ đích Seen_in[|V|] và khởi tạo với giá trị False ;
4      Tạo hàng đợi các đỉnh sẽ duyệt theo chiều đi Q_out, chiều đến Q_in;
5      Q_out ← u ; Seen_out[u] = True; Dist_out = 0 ;          /* đánh dấu u đã được duyệt */
6      Q_in ← v ; Seen_in[v] = True; Dist_in = 0 ;             /* đánh dấu v đã được duyệt */
7      while (Q_out ≠ ∅) && (Q_in ≠ ∅) do
8          if Q_out ≠ ∅ then
9              Dist_out = Dist_out + 1; e ← Q_out.dequeue() ;
10             forall t ∈ G.adjacentNodes(e) do
11                 if !Seen_out[t] then
12                     if t == v then return Dist_out ;
                       /* Nếu t đã được duyệt từ đích, trả về tổng khoản cách từ t đến nguồn và đích    */
13                     if Seen_in[t] then return Dist_out + Dist_in ;
                       /* Nếu không, đánh dấu t đã được duyệt từ nguồn                                  */
14                     Seen_out[t] = True; Q_out ← u ;
15                 end
16             end
17         end
18         if Q_in ≠ ∅ then
19             Dist_in = Dist_in + 1 ; e ← Q_in.dequeue() ;
20             forall t ∈ G.adjacentNodes(e) do
21                 if !Seen_in[t] then
22                     if t == v then return Dist_in ;
                       /* Nếu t đã được duyệt từ nguồn, trả về tổng khoản cách từ t đến nguồn và đích   */
23                     if Seen_out[t] then return Dist_out + Dist_in ;
                       /* Nếu không, đánh dấu t đã được duyệt từ đích                                   */
24                     Seen_in[t] = True; Q_in ← u ;
25                 end
26             end
27         end
28     end
29     return-1 ;
```

Có thể thấy, độ phức tạp tính toán của giải thuật tính khoảng cách trên trong trường hợp tồi nhất vẫn là $O(|V| + |E|)$ do chúng ta vẫn phải duyệt hết toàn bộ số đỉnh và số cạnh của đồ thị.

Trong thực tế, ngoài bài toán tìm đường đi ngắn nhất giữa hai đỉnh trong đồ thị, chúng ta còn gặp một số bài toán biến thể tìm đường đi ngắn nhất như sau:

- **SSSP**: là bài toán tìm đường đi ngắn nhất từ một đỉnh nguồn u đến tất cả các đỉnh còn lại trong đồ thị G (single-source shortest path hay viết tắt là SSSP).

- **SDSP**: là bài toán tìm đường đi ngắn nhất đến đỉnh đích v từ tất cả các đỉnh còn lại trong đồ thị G (single-destination shortest path hay viết tắt là SDSP). Rõ ràng, với đồ thị có hướng, giải bài toán SDSP hoàn toàn có thể dựa vào SSSP bằng cách đổi ngược chiều tất cả các cạnh trong G. Còn đối với đồ thị vô hướng, SDSP lại chính là SSSP.

- **APSP**: là bài toán tìm đường đi ngắn nhất giữa bất kỳ cặp đỉnh u, v nào trong đồ thị

G (all-pairs shortest path hay viết tắt là APSP).

Với bài toán SSSP, chúng ta hoàn toàn có thể giải được thông qua việc duyệt từ đỉnh nguồn u bằng giải thuật BFS đối với kiểu đồ thị không trọng số. Trong trường hợp G là đồ thị có trọng số, giải thuật Dijkstra được sử dụng để giải SSSP.

Việc giải bài toán APSP có thể tiến hành thông qua việc thực hiện tính SSSP đối với tất cả các đỉnh trong G. Lúc đó, nếu đồ thị có trọng số, độ phức tạp của việc giải APSP sử dụng giải thuật Dijkstra đối với tất cả đỉnh trong V sẽ là $O(|V|^2 log(|V|) + |V||E|)$. Còn đối với đồ thị không trọng số, sử dụng BFS cho tất cả các đỉnh sẽ có độ phức tạp là $O(|V|^2 + |V||E|)$. Ngoài ra, chúng ta cũng có thể sử dụng giải thuật Johnson (có độ phức tạp tính toán là $O(|V|^2 log(|V|) + |V||E|)$), hay Floyd–Warshall (với độ phức tạp $O(|V|^3$ để giải bài toán APSP [26].

2.3.2.3 Độ trung tâm

Như đã giới thiệu ở trên, lý thuyết đồ thị hiện được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau, từ các bài toán trong giao vận, tin sinh học, quy hoạch mạng, phân tích mạng Internet... Một trong những ứng dụng điển hình của lý thuyết đồ thị hiện nay chính là trong mô hình hoá các mạng xã hội [106][55][53]. Để phân tích các đồ thị, độ trung tâm (centrality) chính là độ đo quan trọng và được sử dụng rộng rãi hiện nay [2]. Độ trung tâm hướng đến việc tìm các đỉnh (tức thành viên) *"quan trọng"* nhất trong đồ thị. Khi áp dụng khái niệm này cho các lĩnh vực khác nhau, chúng ta có thể tìm được các đỉnh chính trong mạng Internet hay các đỉnh làm lan truyền dịch bệnh khi mô hình hóa bài toán lan bệnh dịch bằng đồ thị. Thực tế, khái niệm "quan trọng" được định nghĩa theo nhiều cách khác nhau khi phân tích đồ thị. Từ đó, cũng có nhiều độ đo trung tâm được đề xuất để làm rõ được tính "quan trọng" khi phân tích mạng đó.

Hiện nay, có bốn loại độ trung tâm hay được sử dụng để phân tích đồ thị mạng nói chung và được định nghĩa như sau:

Định nghĩa 2.20. ***Bậc trung tâm (Degree Centrality)*** *của đỉnh v được định nghĩa như số cạnh liên kết với đỉnh cần xét. Độ đo này được xác định theo công thức sau:*

$$CD(v) = deg(v) : v \in V \tag{2.1}$$

Với đồ thị có hướng, độ đo này còn được phân ra thành bậc trung tâm đi CD_{out} (Outflow Centrality) và bậc trung tâm đến (Inflow Centrality) CD_{in}. Đối với các mạng xã hội, bậc trung tâm này chính là số bạn của một người trên Facebook; hay số người theo dõi (follower) trong Twitter. Đây rõ ràng là độ đo xác định được những người nổi tiếng trực tiếp nhất trên mạng xã hội.

Định nghĩa 2.21. ***Độ trung tâm gần (Closeness Centrality)*** *của một đỉnh v được*

định nghĩa dựa theo công thức do Bavelas đề xuất như sau [14]:

$$CC(v) = \frac{1}{\sum_{u \in V} dst(u, v)} \tag{2.2}$$

với $dst(u, v)$ là khoảng cách ngắn nhất từ đỉnh u đến v.

Từ đó, nếu đỉnh v có độ trung tâm gần càng lớn thì v càng gần với các đỉnh còn lại. Trong luận án này, để tránh xét những đỉnh không thể đi đến được v (có giá trị ∞) đối với đồ thị không kết nối (disconnected graph), chúng ta có thể chỉ cần tính độ đo trung tâm gần CC đối với các đỉnh v trong thành phần liên thông lớn nhất (largest-component) Γ_G của G. Khi đó, nếu đỉnh u không thể đến được từ các đỉnh khác trong G, $CC(u) = 0$.

Bài toán tính CC cho tất cả các đỉnh trong đồ thị G rõ ràng có độ phức tạp tương đương với bài toán SSSP cho tất cả các đỉnh trong G, hay cũng tương đương với bài toán APSP. Điều đó đồng nghĩa độ phức tạp tính toán CC sẽ là $O(|V|^3)$ nếu sử dụng giải thuật Floyd-Warshall và $O(|V|^2 log(|V|) + |V||E|))$ nếu sử dụng giải thuật Johnson. Với những đồ thị có số đỉnh rất lớn, rõ ràng việc tính toán CC cho tất cả các đỉnh mất rất nhiều thời gian tính toán. Một số giải thuật tính xấp xỉ CC đã được đề xuất, chẳng hạn Eppstein và cộng sự đã đề xuất giải thuật tính xấp xỉ ngẫu nghiên với độ phức tạp là $O(\frac{log|V|}{\epsilon^2}(|E| + |V|log|V|))$ [31][32] với lỗi cộng $\epsilon\Delta$ có xác suất $1 - \frac{1}{|V|}$, trong đó $\epsilon > 0$ và Δ là đường kính của G. Sau đó, Okamoto đã phát triển phương pháp để xếp hạng và thu được k-đỉnh có độ trung tâm gần CC cao nhất trong G có độ phức tạp $O((k + |V|^{\frac{2}{3}}.log^{\frac{1}{3}}|V|)(|V|log(|V|) + |E|))$ [82]

Định nghĩa 2.22. ***Độ trung tâm trung gian (Betweenness Centrality)*** *của một đỉnh v được tính bởi công thức sau [27]:*

$$BC(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}, \tag{2.3}$$

với σ_{st} là số đường đi ngắn nhất từ đỉnh s đến t và $\sigma_{st}(v)$ là số đường đi ngắn nhất đó đi qua đỉnh v.

Thực tế, độ đo này chính là số lượng cầu nối trung gian một người đảm nhiệm khi xác lập các quan hệ ngắn nhất giữa những người khác, do Linton Freeman đưa ra năm 1977 [36]. Trong công trình đó, Linton Freeman quan niệm các đỉnh có xác suất cao nằm trên đường đi ngắn nhất giữa hai đỉnh được chọn ngẫu nhiên trong tập đỉnh V thì sẽ có độ trung tâm trung gian nhất (high betweenness).

Khi triển khai, việc tính BC cho tất cả các đỉnh trong đồ thị G tương đương với bài toán APSP. Điều đó đồng nghĩa độ phức tạp tính toán sẽ là $O(|V|^3)$ nếu sử dụng giải thuật Floyd-Warshall; $O(|V|^2 log(|V|) + |V||E|))$ nếu sử dụng giải thuật Johnson. Năm 2001, Brandes [15] có đề xuất kỹ thuật tích luỹ phụ thuộc cho phép giảm độ phức tạp lưu trữ và có độ phức tạp tính toán là $O(|V||E|)$. Như vậy, khi phân tích đồ thị thưa, chúng ta nên sử dụng giải thuật Brandes, còn đối với đồ thị dày, nên sử dụng giải thuật Johnshon hay Floyd-Warshall.

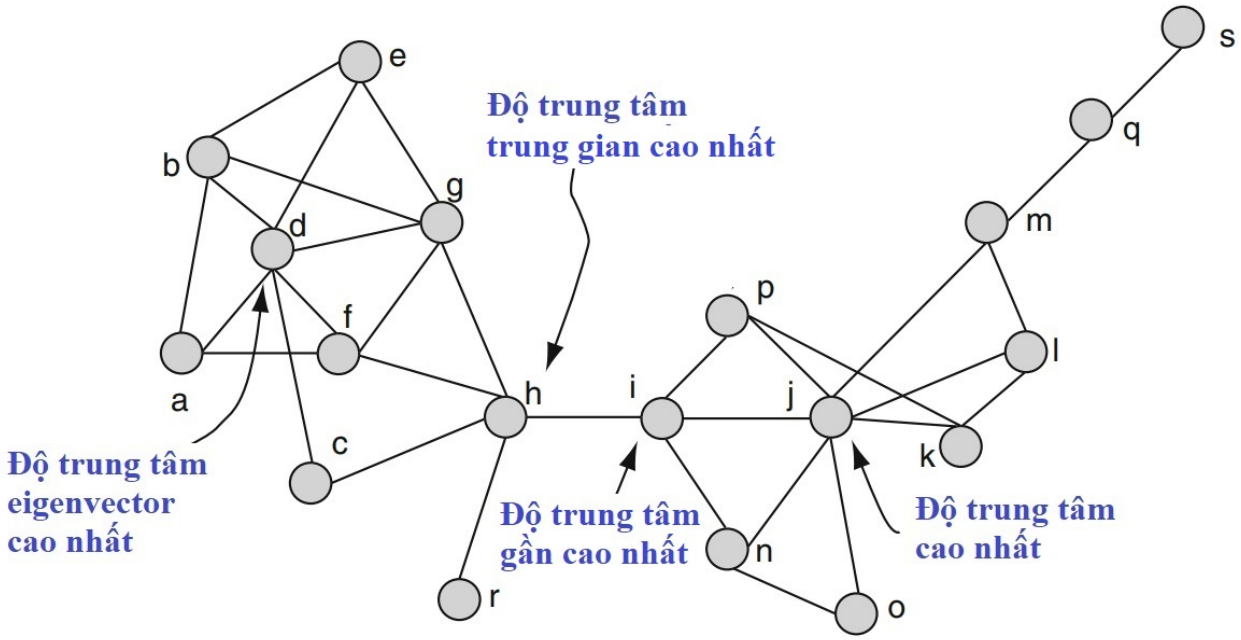
Định nghĩa 2.23. ***Độ trung tâm vector riêng (Eigenvector Centrality)*** *là chỉ dấu để đo mức độ ảnh hưởng của một đỉnh trong đồ thị. Để tính độ đo này, ma trận liền kề* $A = (a_{u,v})$ *được sử dụng để biểu diễn mạng* G*:* $a_{u,v} = 1$ *nếu* u *kết nối với* v *và nếu không* $a_{u,v} = 0$*. Độ trung tâm vector riêng* x *của* v *được định nghĩa như sau:*

$$x_v = \frac{1}{\lambda} \sum_{t \in M(v)} x_t = \frac{1}{\lambda} \sum_{t \in G} a_{v,t} x_t, \tag{2.4}$$

với $M(v)$ *là tập các đỉnh liền kề của* v *và* λ *là hằng số. Theo dạng ma trận, chúng ta có:* $\lambda x = xA$*.*

Như vậy, độ trung tâm vector riêng cho phép gán điểm số tương đối cho tất cả các đỉnh trong mạng dựa trên khái niệm kết nối với những đỉnh có điểm cao thì đóng góp lớn hơn cho đỉnh đang xét so với kết nối với các đỉnh điểm thấp[54]. Các biến thể của độ trung tâm Eigenvector có thể kể đến như độ trung tâm Katz (Katz Centrality) và xếp hạng trang (Page Rank) của Google.

Ví dụ sau đây minh hoạ các độ đo trung tâm đã nêu ở trên [84]:



Hình 2.10: Một số độ đo trung tâm điển hình trên đồ thị

Trong đồ thị G ở hình 2.10 này, đỉnh j có $CD[j] = 7$ là độ trung tâm cao nhất trong G. Tương tự, độ trung tâm gần $CC[i] = 0,461538$, độ trung tâm trung gian $BC[h] = 176$ cũng là lớn nhất. Ngoài ra, độ trung tâm vector riêng tại đỉnh d cũng đạt giá trị lớn nhất $x_d = 0,4647$ trong số các đỉnh còn lại trong đồ thị này.

Trong thực tế, các độ đo trung tâm này có thể có các biến thể khác với cách cài đặt sử dụng giải thuật khác nhau [79].

Trong luận án này, với mục tiêu đặt ra, chúng tôi sẽ chú trọng đến việc nâng cao hiệu năng thi hành các phép toán đồng thời và tính một số độ đo trung tâm đối với các đồ thị có quy mô lớn cả về số đỉnh lẫn số cạnh (lớn hơn triệu đỉnh/cạnh). Hiển nhiên, các phương thức nâng cao hiệu năng của chúng tôi cũng phải đảm bảo hoàn toàn áp dụng được với các đồ thị quy mô trung bình và nhỏ. Ý tưởng chính trong việc nâng cao hiệu năng này một phần được dựa trên phương pháp thi hành song song. Như đã trình bày ở mục 1.2, để giải quyết các thách thức liên quan đến phân tích đồ thị quy mô lớn, tất cả các cách tiếp cận hiện nay đều sử dụng phương pháp tính toán song song. Từ đó, phần lý thuyết liên quan đến tính toán song song sẽ được trình bày sơ lược trong mục 2.4 dưới đây.

2.3.3 Mật độ đồ thị

Định nghĩa 2.24. *Độ đo mật độ (density) được định nghĩa dựa trên độ đo số cạnh của đồ thị trên tổng số cạnh tối đa có thể có đối với tập đỉnh của đồ thị đó [109]. Công thức tính mật độ đồ thị D được định nghĩa như sau:*

$$D = \frac{2|E|}{|V|(|V|-1)} \tag{2.5}$$

với G là đồ thị đơn vô hướng. Trong trường hợp G là đồ thị đơn có hướng, D được tính như sau:

$$D = \frac{|E|}{|V|(|V|-1)} \tag{2.6}$$

Như vậy, với đồ thị đầy đủ vô hướng thì $D = 1$ và với một số trường hợp đặc biệt, mật độ đồ thị có thể có giá trị lớn hơn 1, chẳng hạn với đồ thị có vòng lặp tại các đỉnh (solving loops). Mật độ đồ thị cho phép chúng ta phân biệt được đồ thị dày (dense graph) và đồ thị thưa (sparse graph) tương ứng với giá trị D tiệm cận giá trị 1 hay khi có giá trị rất nhỏ.

2.3.4 Phân cụm đồ thị

Định nghĩa 2.25. *Phân cụm đồ thị (graph clustering) là quá trình phân chia đồ thị thành nhiều nhóm mà mỗi đỉnh trong nhóm có độ tương tự (chẳng hạn độ trung tâm gần) lớn hơn so với các đỉnh ở nhóm khác [93].*

Để phục vụ cho bài toán phân cụm đồ thị, hệ số phân cụm (Clustering Coefficient) đã được đề xuất [94]. Đây là độ đo mức độ mà các đỉnh trong đồ thị có xu hướng nhóm lại được với nhau. Thực tế đối với các đồ thị mạng xã hội, các thành viên thường hình thành các nhóm liên kết tương đối chặt với nhau với các quan hệ xã hội thông thường. Từ đó, hệ số phân cụm cho phép đánh giá mức độ hình thành nhóm của một số các đỉnh trong đồ thị.

2.4 Tính toán song song

Định nghĩa 2.26. *Tính toán song song (parallel computing) là kiểu tính toán trong đó nhiều phép tính (hoặc việc thi hành nhiều tiến trình) được tiến hành một cách đồng thời dựa trên nguyên tắc những bài toán lớn đều có thể chia thành nhiều bài toán nhỏ hơn có thể tiến hành đồng thời [42].*

Tính toán song song đã được sử dụng rộng rãi từ nhiều năm qua với mục tiêu rút ngắn thời gian thi hành của các bài toán cần tính toán nhanh [4]. Có nhiều hình thức thể hiện của tính toán song song: (i) song song mức bit, (ii) mức lệnh trong CPU, (iii) mức dữ liệu, hay (iv) mức tác vụ [1]. Hiện nay, tính toán song song đã trở thành một dạng thức (paradigm) thống trị trong lĩnh vực kiến trúc máy tính dưới dạng các bộ vi xử lý đa nhân (multi-core) [102]. Hầu hết các máy tính hiện nay đều có nhiều nhân và các hệ điều hành khai thác hạ tầng tính toán trên các máy tính đó đều vừa được xây dựng dựa trên các phương pháp tính toán song song, vừa cung cấp các thư viện cho phép triển khai được các phép tính đồng thời.

Các phép toán khi được triển khai theo mô hình tính toán song song thường được gọi các các phép toán tương tranh [5]. Việc thi hành các phép toán tương tranh nói chung cần phải có phương pháp phối hợp giữa các phép toán đó để việc thao tác với những vùng dữ liệu chung hay sử dụng các kết quả lẫn nhau được tiến hành một cách nhất quán, đảm bảo sự tin cậy của kết quả tính toán [3]. Việc các hệ thống máy tính ngày nay đều hỗ trợ khả năng đa nhiệm, đa người dùng chính là dựa vào những thành quả của việc giải quyết các bài toán tương tranh và khai thác được thế mạnh tính toán song song trên các bộ vi xử lý hiện đại.

Việc đánh giá hiệu năng của các mô hình tính toán song song hiện nay thường sử dụng luật Amdahl (Amdahl's Law) [3]. Amdahl cho rằng hệ số tăng tốc của một chương trình song song được xác lập dựa vào tỷ lệ mã (code) P của chương trình đó được song song hoá:

$$SpeedUp = \frac{1}{1 - P} \tag{2.7}$$

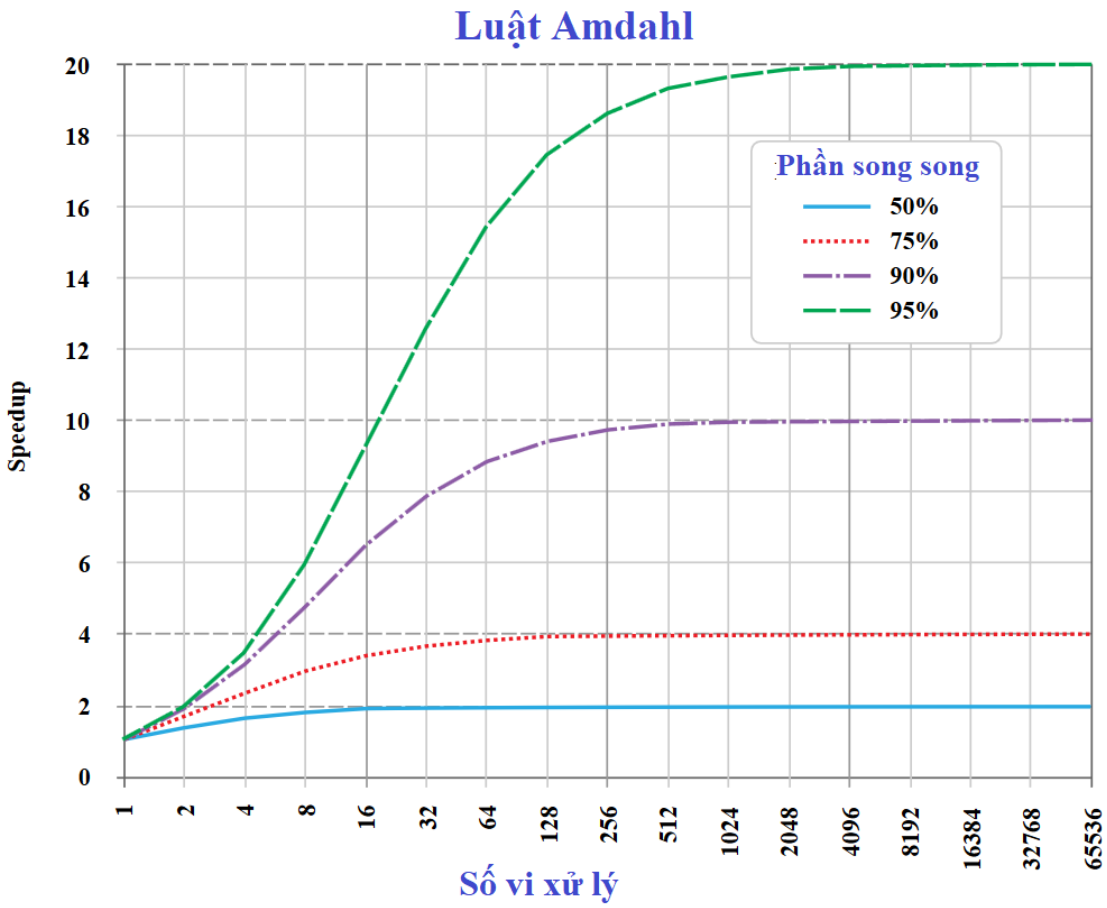
Từ đó, nếu không có phần nào trong chương trình được song song, hệ số tăng tốc $SpeedUp = 1$. Nếu tất cả mã của chương trình được song song hoá thì $SpeedUp = \infty$. Tuy nhiên đây chỉ là trường hợp phi thực tế vì không tồn tại chương trình nào mà tất cả mã đều song song được.

Khi xét thêm số lượng CPU tham gia vào tính toán song song, hệ số này được xác định bằng biểu thức sau:

$$SpeedUp = \frac{1}{S + \frac{P}{N}} \tag{2.8}$$

trong đó, P là tỷ lệ mã song song, N là số lượng CPU và S là tỷ lệ mã tuần tự.

Tương quan của các tham số này được minh hoạ cụ thể như hình sau:



Hình 2.11: Ảnh hưởng của số CPU và tỷ lệ đoạn mã được song song đến hệ số tăng tốc

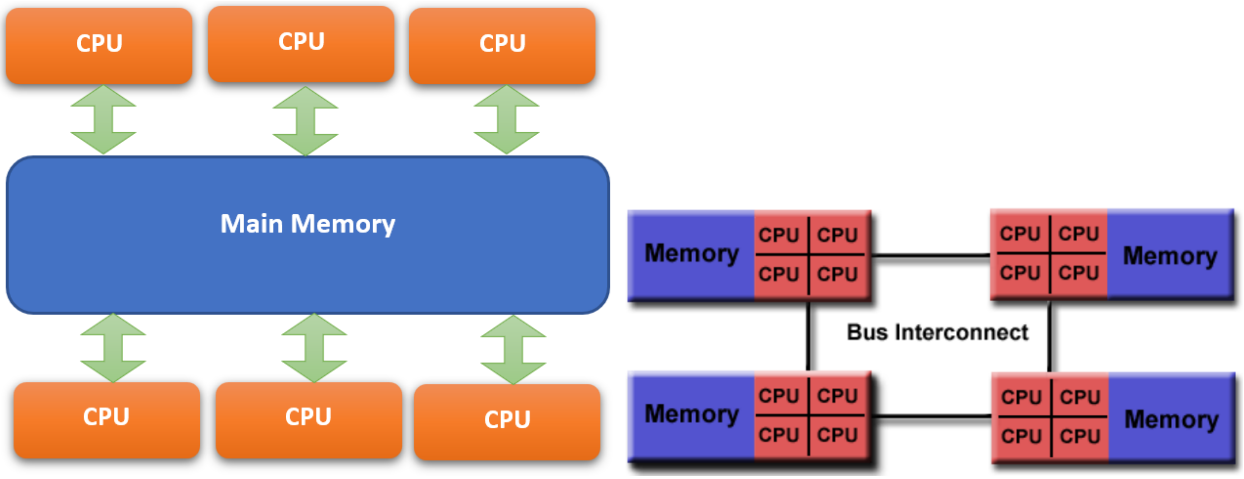
2.4.1 Kiến trúc hệ thống tính toán song song

Các phương pháp tính toán song song hiện nay đều dựa trên các hạ tầng tính toán cho phép sử dụng phối ghép năng lực tính toán đơn lẻ ở các bộ vi xử lý (Central Processing Unit - CPU) để hình thành hệ thống tính toán có thể xử lý được nhiều phép toán hơn. Các hệ thống phần cứng đó được chia thành các kiến trúc điển hình sau:

2.4.1.1 Kiến trúc bộ nhớ chia sẻ

Đây là kiến trúc phần cứng được sử dụng rộng rãi trong các hệ thống tính toán hiện nay. Kiến trúc này được xây dựng trên cách tiếp cận chung là tất cả các CPU đều có thể truy cập sử dụng bộ nhớ chính như là không gian địa chỉ toàn cục. Các bộ vi xử lý CPU có thể hoạt động độc lập với bộ nhớ đệm cache riêng, nhưng đều chia sẻ không gian bộ nhớ. Điều đó dẫn đến việc thay đổi từ nhớ trong không gian bộ nhớ bởi một CPU sẽ ảnh hưởng đến toàn bộ các CPU truy cập đến từ nhớ đó.

Kiến trúc bộ nhớ chia sẻ thường được phân thành hai loại là bộ nhớ chia sẻ truy cập thống nhất (Uniform Memory Access - UMA) và bộ nhớ chia sẻ truy cập không thống nhất (Non-Uniform Memory Access - NUMA).



(a) Uniform Memory Access (b) Non-Uniform Memory Access

Hình 2.12: Kiến trúc hệ thống tính toán song song dựa trên bộ nhớ chia sẻ

Đối với kiến trúc dựa trên truy cập thống nhất UMA, các CPUs phải đồng nhất và việc tham chiếu đến không gian bộ nhớ chính là giống nhau. Kiến trúc này được sử dụng chủ yếu trong các hệ thống máy tính SMP (Symmetric Multiprocessor), đôi khi còn gọi là kiến trúc CC-UMA (Cache Coherent UMA). Thuật ngữ "gắn kết bộ nhớ đệm - cache coherency" sẽ đảm bảo khi một CPU cập nhật từ nhớ trong bộ nhớ, tất cả các CPUs sử dụng từ nhớ đó sẽ có bộ nhớ cache được cập nhật giá trị mới của từ nhớ đó. Đây cũng là lý do mà công nghệ này được thực hiện ở phần cứng trong việc xây dựng các CPU.

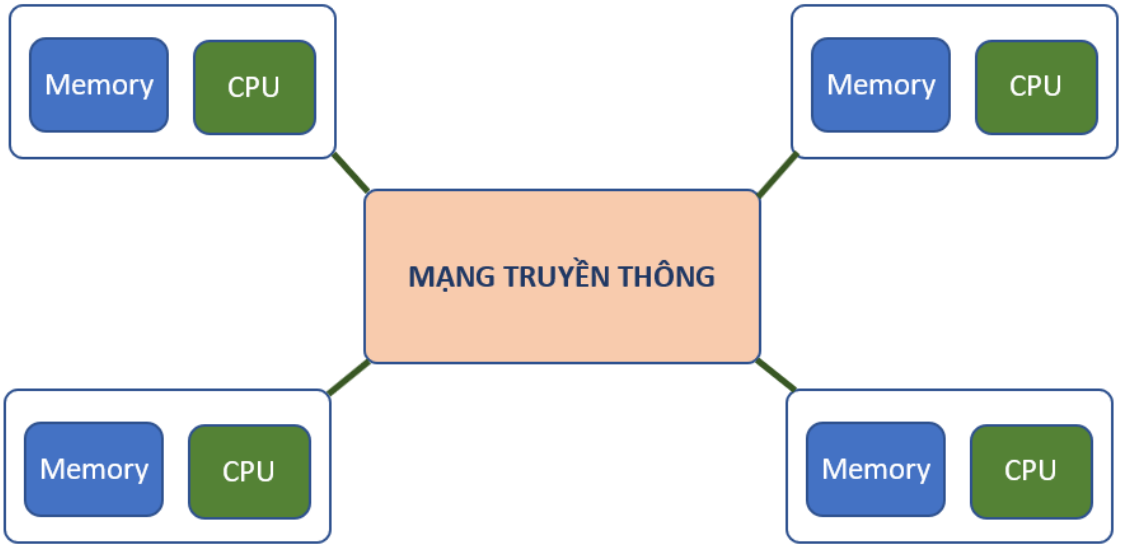
Còn với kiến trúc chia sẻ bộ nhớ NUMA, không cần thiết phải đảm bảo các CPUs phải đồng nhất cũng như có cùng thời gian tham chiếu đến bộ nhớ toàn cục. Đây là kiến trúc hay được xây dựng dựa trên việc kết nối hai hoặc nhiều hệ thống SMP và cho phép CPU từ SMP này có thể truy cập đến bộ nhớ của SMP khác. Cơ chế "cache coherency" cũng phải được cài đặt trong các hệ thống đó, và vì vậy, kiến trúc này còn đôi khi được gọi là kiến trúc CC-NUMA.

Như vậy, kiến trúc chia sẻ bộ nhớ UMA và NUMA mang lại khả năng có thể tính toán song song trên cùng không gian bộ nhớ được chia sẻ chung. Tuy nhiên, kiến trúc này cũng thiếu sự linh hoạt khi cần mở rộng thêm CPUs và càng nhiều CPUs thì hệ thống liên kết cũng như cơ chế đồng bộ dữ liệu trong bộ nhớ đệm lại càng phức tạp.

2.4.1.2 Kiến trúc bộ nhớ phân tán

Kiến trúc bộ nhớ phân tán là kiến trúc tính toán phân tán sử dụng mạng truyền thông để kết nối bộ nhớ liên bộ xử lý. Trong kiến trúc này, mỗi bộ vi xử lý có riêng không gian bộ nhớ cục bộ và không được ánh xạ (map) vào các CPU khác (khác với việc sử dụng chung không gian bộ nhớ toàn cục như kiến trúc bộ nhớ chia sẻ). Từ đó, việc cập nhật bộ nhớ của một CPU không ảnh hưởng đến quá trình tính toán tại các CPU khác và không cần cài đặt cơ chế cache coherency.

Với kiến trúc này, việc tương tác giữa các CPU trong quá trình tính toán song song sẽ được thực hiện thông qua những phương thức truyền thông mạng, thông thường dựa vào truyền các messages trên mạng cục bộ ethernet. Chương trình thi hành tính toán song song phải đảm nhiệm việc truyền thông cũng như đồng bộ dữ liệu giữa các CPUs với nhau. Kiến trúc này được minh hoạ như hình dưới đây:

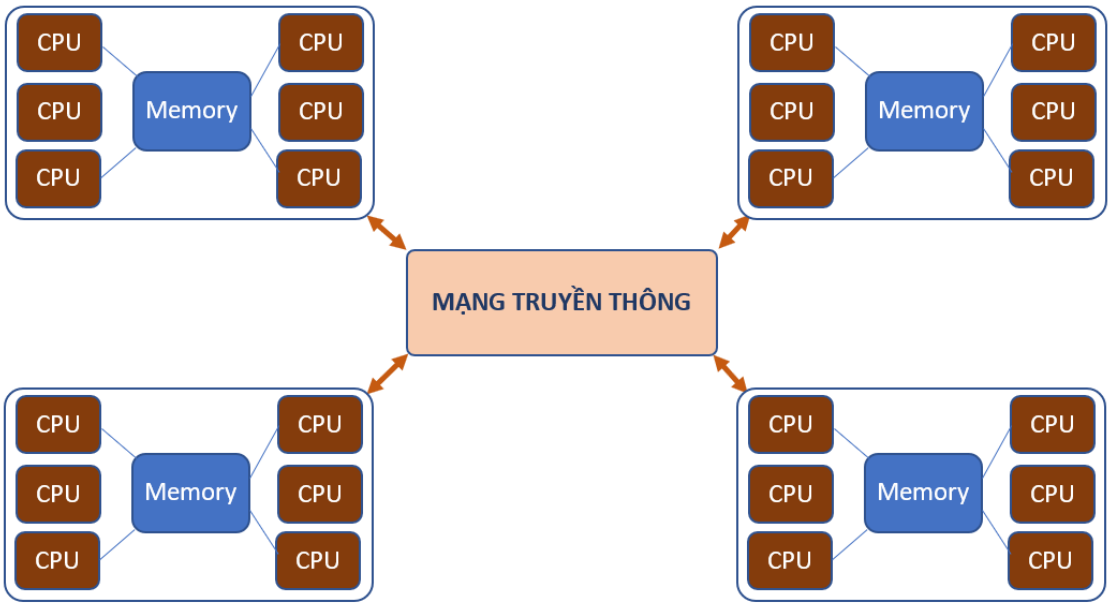


Hình 2.13: Kiến trúc hệ thống tính toán song song dựa trên bộ nhớ phân tán

Đối ngược với kiến trúc chia sẻ bộ nhớ, ưu điểm chính của kiến trúc này chính là tính khả mở với số lượng lớn CPU cũng như dung lượng bộ nhớ. Ưu điểm nữa của kiến trúc này là mỗi CPU có môi trường tính độc lập, không ảnh hưởng đến các CPU còn lại. Về nhược điểm, hiệu năng tính toán song song của kiến trúc này phụ thuộc rất nhiều vào mạng truyền thông giữa các CPU.

2.4.1.3 Kiến trúc bộ nhớ lai chia sẻ-phân tán

Kiến trúc bộ nhớ lai chia sẻ-phân tán là kiến trúc kết hợp cả hai kiến trúc đã nêu trên. Các hệ thống tính toán hiệu năng cao hiện nay trên thế giới đều sử dụng kiến trúc lai này. Kiến trúc lai được minh hoạ như hình sau:



Hình 2.14: Kiến trúc hệ thống tính toán song song dựa trên bộ nhớ lai chia sẻ-phân tán

Thành phần chia sẻ trong kiến trúc này cũng được mở rộng không chỉ là chia sẻ bộ nhớ mà còn cả các bộ xử lý đồ hoạ (Graphics Processing Units - GPU). Thành phần phân tán là mạng của các máy chia sẻ bộ nhớ hay GPU. Rõ ràng kiến trúc này cho phép tận dụng được cả hai ưu điểm của hai kiến trúc nêu trên, tăng độ linh hoạt khả mở CPU lẫn khai thác triệt để tính toán song song trên cùng không gian bộ nhớ chia sẻ.

2.4.2 Mô hình lập trình song song

Mô hình lập trình song song (parallel programming models) là mô hình để xây dựng các hệ thống tính toán với các kiến trúc đã trình bày ở mục trên. Hiện nay, các giải pháp tính toán song song thường được xây dựng dựa vào một số mô hình lập trình song song sau [5]:

- Bộ nhớ chia sẻ (không sử dụng luồng): trong mô hình lập trình đơn giản này, quá trình tính toán song song được thi hành thông qua các tiến trình (với tài nguyên bộ nhớ, CPU riêng) nhưng có vùng nhớ chia sẻ chung để đọc/ghi theo cơ chế không đồng bộ. Do đó, các cơ chế như sử dụng khoá, semaphores... cần phải được sử dụng để kiểm soát tương tranh, tránh khoá chết... trong việc truy xuất đến dữ liệu chung giữa các tiến trình.

- Sử dụng luồng (threads): đây là mô hình lập trình sử dụng bộ nhớ chia sẻ, tuy nhiên việc tính toán song song được triển khai trong một tiến trình bao hàm nhiều luồng (threads) thi hành tương tranh. Mỗi luồng sẽ có dữ liệu cục bộ nhưng đều chia sẻ toàn bộ tài nguyên của tiến trình cha. Thông thường, mỗi luồng sẽ được giao phó để thi hành một hàm và việc tương tác giữa các luồng thông qua không gian bộ nhớ toàn

cục. Điều này cũng dẫn đến phải có cơ chế đồng bộ để đảm bảo việc cập nhật dữ liệu chung giữa các luồng.

Về mặt lập trình, mô hình sử dụng luồng này có thể được thực hiện hoặc (i) sử dụng những thư viện cung cấp các hàm để cài đặt chương trình song song (điển hình là POSIX Threads [10], Microsoft Threads [73], Java/Python threads [1], CUDA threads[2]), hoặc (ii) sử dụng các chỉ thị đã được nhúng trong các ngôn ngữ lập trình để xác lập đoạn mã khi nào song song, khi nào tuần tự (điển hình là OpenMP[9], CilkPlus [49]).

- Bộ nhớ phân tán với phương pháp truyền thông điệp (message passing): Trong mô hình lập trình này, các tác vụ sẽ có không gian bộ nhớ riêng và có thể được thi hành trên một máy tính hoặc trên nhiều máy khác nhau. Quá trình trao đổi dữ liệu giữa chúng sẽ được tiến hành dựa trên gửi và nhận thông điệp. Về mặt lập trình, cài đặt cơ chế truyền thông điệp thường được đóng gói trong một thư viện lập trình. Lập trình viên phải sử dụng các hàm trong thư viện đó để đảm nhiệm cài đặt cơ chế song song cho bài toán cần giải quyết. Thư viện MPI (Message Passing Interface) hiện là thư viện được sử dụng rộng rãi nhất hiện nay trong mô hình lập trình này với ba chuẩn là MPI-1, MPI-2 và MPI-3 [8].

- Song song dữ liệu: là mô hình lập trình song song sử dụng không gian bộ nhớ toàn cục nhưng các công việc song song tập trung thao tác với các tập dữ liệu có cấu trúc chung, chẳng hạn như mảng hay khối dữ liệu. Mỗi công việc tiến hành trên một vùng khác nhau của dữ liệu toàn cục đó. Mô hình này còn có tên gọi là mô hình phân mảnh địa chỉ toàn cục (Partitioned Global Address Space - PGAS). Với kiến trúc bộ nhớ chia sẻ, tất cả công việc có thể truy cập cấu trúc dữ liệu qua bộ nhớ toàn cục. Với kiến trúc bộ nhớ phân tán, cấu trúc dữ liệu này có thể được phân chia một cách logic hoặc/và vật lý giữa các công việc.

  Hiện nay, mô hình này cũng có nhiều cài đặt khác nhau được sử dụng trong các ngôn ngữ lập trình khác nhau. Chẳng hạn như thư viện *CoArray* của Fortran[3]; *Unified Parallel C (UPC)* nhúng trong ngôn ngữ C để lập trình kiểu SPMD (Single Program Multiple Data: đơn chương trình, đa dữ liệu)[4]; Mảng toàn cục; X10, Chapel... [5].

- Mô hình lai: là mô hình lập trình sử dụng nhiều kiểu mô hình trên kết hợp cùng với nhau. Hiện nay, một số kiểu lai hay được sử dụng có thể kể đến như kết hợp MPI với mô hình luồng OpenMP; MPI với mô hình luồng sử dụng các bộ xử lý đồ hoạ GPU (CUDA chẳng hạn); MPI với Pthreads; ...

[1]Tham khảo tại `https://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html` và `https://en.wikibooks.org/wiki/Python_Programming/Threading`

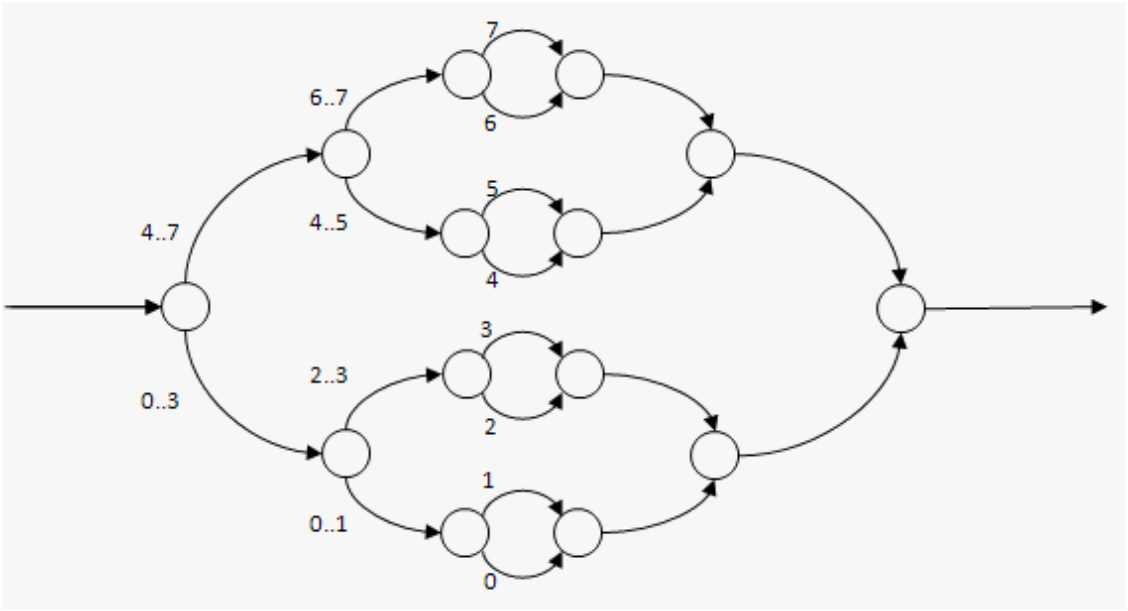
[2]Lập trình luồng với Cuda `http://cuda.ce.rit.edu/cuda_overview/cuda_overview.htm`

[3]Tham khảo thêm tại `https://en.wikipedia.org/wiki/Coarray_Fortran`

[4]Tham khảo chi tiết tại `http://upc.lbl.gov/`

Nhìn chung, các mô hình lập trình độc lập so với kiến trúc bộ nhớ, chẳng hạn như mô hình lập trình bộ nhớ chia sẻ có thể triển khai trên các máy tính có kiến trúc bộ nhớ phân tán và ngược lại. Trong số các mô hình trên, không có mô hình nào là "tốt" cho tất cả các trường hợp mà mỗi mô hình hiệu quả với từng bài toán cụ thể [102].

Trong các giải pháp song song hoá được chúng tôi tiến hành thực hiện trong luận án này, mô hình lập trình song song sử dụng đa luồng trên kiến trúc đa CPU, đa lõi được chú trọng thực hiện. Các giải pháp song song sẽ được cài đặt dựa trên bộ thư viện CilkPlus do Intel xây dựng [49].



Hình 2.15: Mô hình xử lý song song trong CilkPlus

Việc lựa chọn CilkPlus cũng đã được chúng tôi phân tích bằng cả thực nghiệm lẫn tham khảo các công trình liên quan. Trong số các giải pháp song song hoá hay được sử dụng hiện nay như CilkPlus, OpenMP[5], và Pthread[6], công trình của Leist và Gilman [59] thông qua những thực nghiệm đã minh chứng bộ thư viện CilkPlus cho hệ số tăng tốc (speedup) tốt hơn so với OpenMP và Pthread.

2.4.3 Một số bài toán song song điển hình

Với những hạ tầng tính toán có hiệu năng ngày càng cao hiện nay, việc áp dụng tính toán song song để khai thác hết năng lực phần cứng cũng ngày càng được quan tâm. Tính toán song song hiện được sử dụng trong phạm vi rất rộng, từ sinh học (phân tích chuỗi DNA chẳng hạn) đến kinh tế (toán tài chính). Các dạng bài toán thường gặp cần ứng dụng tính toán song song là [4]:

[5] `http://openmp.org/wp/`
[6] `https://computing.llnl.gov/tutorials/pthreads/`

- Đại số tuyến tính dày đặc, thưa.
- Phương pháp quang phổ (chẳng hạn biến đổi Fourier nhanh của Cooley–Tukey).
- Bài toán n-body (như mô phỏng Barnes–Hut).
- Bài toán mạng lưới cấu trúc (như Phương pháp lưới Boltzmann), phi cấu trúc (như trong phân tích phần tử hữu hạn).
- Phương pháp mô phỏng Monte Carlo.
- Tổ hợp logic (như kỹ thuật mật mã kiểu brute-force).
- Duyệt đồ thị (như thuật toán tìm đường đi ngắn nhất).
- Quy hoạch động.
- Phương pháp nhánh và cận...

2.5 Kết chương 2

Trong chương này chúng tôi đã trình bày những kiến thức nền tảng về đồ thị gắn với các nghiên cứu chuyên sâu trong luận án. Một trong những xu hướng điển hình trong việc quản lý dữ liệu quy mô lớn hiện nay là áp dụng lý thuyết đồ thị để nâng cao hiệu năng xử lý. Theo cách tiếp cận đó, thực thể sẽ được biểu diễn dưới dạng đỉnh còn quan hệ giữa các thực thể được thể hiện dưới các cạnh. Từ đó, các khái niệm, kiểu đồ thị cũng đã được chúng tôi trình bày tường minh trong chương này trước khi đặc tả chi tiết các phương pháp biểu diễn dữ liệu đồ thị.

Từ các vấn đề đó, chúng tôi đã đi sâu giới thiệu một số các phép toán chính trong đồ thị với hai loại chính: các phép toán duyệt đồ thị và các phép toán phân tích đồ thị. Các phép toán này chính là nền tảng để cho phép duyệt hiệu quả đồ thị, hình thành các mô tơ xử lý truy vấn của các bài toán thực tế ứng dụng lý thuyết đồ thị.

Trong luận án, tính toán song song sẽ được chú trọng nghiên cứu ứng dụng để nâng cao hiệu năng xử lý các phép toán trên đồ thị. Các khái niệm cơ bản liên quan đến lĩnh vực này cũng được trình bày trong chương này và sẽ là cơ sở để xây dựng các phương thức cải thiện hiệu năng các phép toán trên đồ thị.

Từ những nền tảng lý thuyết đó, trong các chương kế tiếp, chúng tôi sẽ đi sâu đặc tả các kết quả và đóng góp chính của luận án liên quan đến việc cải thiện hiệu năng tính toán dựa theo cách tiếp cận sử dụng biểu diễn dữ liệu đồ thị phù hợp và nâng cao hiệu năng thi hành các phép toán đồng thời trên đồ thị có quy mô lớn. Phạm vi nghiên cứu của luận án chỉ tập trung vào đồ thị không trọng số, có quy mô lớn ở mức hàng triệu đỉnh/cạnh (dĩ nhiên, các

kết quả nghiên cứu của luận án hoàn toàn áp dụng được với quy mô trung bình và nhỏ). Quy mô siêu lớn (hàng tỷ đỉnh/cạnh) chưa được xét trong phạm vi luận án này.

Từ đó, chương 3 kế tiếp sẽ trình bày về tối ưu hoá các phép toán đồng thời trên đồ thị động, quy mô lớn. Chương 4 sẽ chú trọng đến các phép toán tính độ trung tâm phục vụ phân tích trên các đồ thị quy mô lớn (chẳng hạn như các đồ thị biểu diễn mạng xã hội).

Chương 3

TỐI ƯU HOÁ TRUY VẤN KHOẢNG CÁCH NGẮN NHẤT TRÊN ĐỒ THỊ ĐỘNG

3.1 Giới thiệu

Ngày nay, chúng ta sống trong xã hội mà khái niệm kết nối hình thành "mạng" trở nên quan trọng và thể hiện ở khắp mọi lĩnh vực nói chung. Các mạng này, chẳng hạn như mạng xã hội, mạng sinh học, mạng phân phối nội dung, mạng lưới giao thông, mạng ngữ nghĩa... đều sinh dữ liệu với tốc độ nhanh và với quy mô lớn. Để xử lý thách thức dữ liệu lớn, cách tiếp cận mô hình hoá dữ liệu dựa trên đồ thị được coi là một trong những phương pháp tự nhiên và hiệu quả nhất hiện nay [75] [74]. Với việc sử dụng lý thuyết đồ thị, các đỉnh đồ thị thường được sử dụng để biểu diễn các thực thể và các cạnh được thiết kế để biểu thị tương tác, quan hệ giữa chúng. Trong đồ thị, bài toán tìm đường đi tối ưu (ngắn nhất) là bài toán xác lập đường đi giữa hai đỉnh sao cho chi phí của đường đi đó là thấp nhất (chi phí này được tính dựa trên trọng số gắn với từng cạnh trong trường hợp xét đồ thị có trọng số; với đồ thị không trọng số, chi phí này được tính bằng số cạnh cấu thành nên đường đi đó). Đây là một bài toán tối ưu hóa cơ bản và đã được nghiên cứu từ nhiều năm qua với nhiều ứng dụng thực tế: tối ưu hoá lược đồ định tuyến trong mạng Internet; tìm đường đi ngắn nhất trong mạng lưới giao thông; tìm kiếm thông tin trong các trang Web; xác lập các quan hệ giữa các thành viên tham gia mạng xã hội...[97]. Như đã giới thiệu trong chương 1, bài toán tìm đường đi tối ưu hiện đã được giải quyết nhờ sử dụng giải thuật BFS với đồ thị không trọng số, hoặc giải thuật Dijkstra với đồ thị có trọng số. Tuy nhiên, khi triển khai thực tế một đồ thị vừa quy mô lớn vừa có tính động, thay đổi liên tục theo thời gian như mạng xã hội, việc xử lý các truy vấn tìm đường đi tối ưu lại trở thành một thách thức lớn [104].

Trong chương này, chúng tôi trình bày giải pháp của mình để cải thiện hiệu suất xử lý các truy vấn đồng thời trên đồ thị có hướng, không trọng số với quy mô lớn và có tính thay

đổi nhanh. Chiến lược của chúng tôi dựa trên các ý tưởng: (i) cấu trúc dữ liệu thích hợp, (ii) giảm thiểu không gian tìm kiếm, và (iii) thi hành song song hiệu quả.

Các nội dung trong chương này được tổ chức như sau: trong mục 3.2 chúng tôi sẽ trình bày đặc tả chi tiết bài toán tối ưu hoá các truy vấn khoảng cách ngắn nhất trên đồ thị động quy mô lớn. Sau đó, trong các mục 3.3, 3.4 và 3.5 chúng tôi sẽ trình bày lần lượt ba giải pháp để giải bài toán tối ưu hoá truy vấn đã đặt ra. Từ các giải pháp đó, chúng tôi đã tiến hành thực nghiệm, so sánh, đánh giá với một số giải pháp khác đã được công bố trong mục 3.6. Mục cuối cùng của chương này tóm lược lại những kết quả và đóng góp chính. Các kết quả của chương 3 cũng được thể hiện trong các công bố sau:

- [**DPH1**] **Phuong-Hanh DU**, Hai-Dang Pham, Ngoc-Hoa Nguyen, "Optimizing the Shortest Path Query on Large-Scale Dynamic Directed Graph", *BDCAT '16 Proceedings of the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, pp210-216, 2016. (SCOPUS, WoS)

- [**DPH2**] **Phuong-Hanh DU**, Hai-Dang Pham, Ngoc-Hoa Nguyen, "An Efficient Parallel Method for Performing Concurrent Operations on Social Networks", *Computational Collective Intelligence*, Volume 10448 of the series Lecture Notes in Computer Science, Springer, pp 148-159, 2017. (SCOPUS, WoS)

- [**DPH3**] **Phuong-Hanh DU**, Hai-Dang Pham, Ngoc-Hoa Nguyen, "An Efficient Parallel Method for Optimizing Concurrent Operations on Social Networks", *Transactions on Computational Collective Intelligence*. Lecture Notes in Computer Science (Q2), vol 10840, no XXIX, pp. 182-199. Springer 2018, ISSN 2190-9288 (0302-9743). (SCOPUS, WoS)

3.2 Đặc tả bài toán

Xét đồ thị $G = (V, E)$ là đồ thị có tập đỉnh V và E lớn, từ vài triệu đỉnh đến hàng tỷ đỉnh, từ vài triệu cạnh đến vài trăm tỷ cạnh (chẳng hạn mạng xã hội Facebook hiện đã có hơn 2,3 tỷ thành viên; từ năm 2012 đã có hơn 140 tỷ quan hệ [1]). Giả sử G liên tục có những thay đổi (thêm/bớt) về số lượng thực thể V (tức số lượng đỉnh) và thay đổi về số lượng quan hệ (tức số lượng cạnh), chẳng hạn như với trường hợp mô hình hoá các mạng xã hội hiên nay. Các quan hệ E giữa các thực thể trong G được mô hình hoá không xét đến trọng số và có hướng. Tức mỗi cặp $(u, v) \in E$ xác lập một quan hệ u có ảnh hưởng v, nhưng chưa xác lập chiều ngược lại. Việc mô hình hoá này rõ ràng hoàn toàn có thể mở rộng để giải quyết các bài toán tổng thể như với quan hệ vô hướng thì chỉ cần bổ sung thêm cạnh $(v, u) \in E$;

[1] Xem chi tiết tại trang `https://mashable.com/2012/10/05/the-most-important-facebook-number-140-billion/`

quan hệ có trọng số thì bổ sung liên tiếp các đỉnh trung gian tương ứng với trọng số chẳng hạn.

Trong luận án này, với ngữ cảnh nêu trên, chúng tôi quan tâm đến bài toán xử lý các truy vấn đồng thời, tìm cách xác lập quan hệ tối ưu nhất để thành viên u ảnh hưởng đến thành viên v trong đồ thị G có khả năng có thay đổi liên tục. Đây chính là bài toán hay gặp trên các mạng xã hội mà ở đó, số lượng thành viên (tức các đỉnh) cũng như số lượng quan hệ (các cạnh) có sự biến động liên tục (thêm mới, tự huỷ...). Ngay trong quá trình đồ thị đang thay đổi đó, có thể có rất nhiều truy vấn đánh giá mức độ quan hệ giữa hai thành viên thông qua việc xác định khoảng cách ngắn nhất giữa chúng.

3.2.1 Mô hình dữ liệu và truy vấn

Như chương 2 đã trình bày về lý thuyết đồ thị và các phương pháp biểu diễn dữ liệu đồ thị điển hình, với đồ thị có hướng không trọng số $G = (V, E)$, mỗi đỉnh của đồ thị sẽ được định danh bởi một số tự nhiên. Điều đó dẫn đến tập đỉnh v sẽ được ánh xạ trong khoảng giá trị $[0..(|V| - 1)]$. Còn đối với các quan hệ E, có ba cách để biểu diễn (như trong mục 2.2): (i) danh sách cạnh, (ii) ma trận liền kề và (iii) danh sách liền kề. Với giả thiết xét các đồ thị có quy mô lớn, việc biểu diễn theo kiểu ma trận liền kề sẽ không hiệu quả do vượt quá dung lượng không gian bộ nhớ chính trong các hệ thống tính toán thông thường (thực vậy, với G có 1 triệu đỉnh thì ma trận liền kề mà mỗi phần tử được cấp phát 1 byte thì không gian bộ nhớ cần cho ma trận liền kề sẽ là 10^6 x 10^6 x 1 = 10^{12}B = 10TB; rõ ràng không thể đưa trọn vẹn vào bộ nhớ chính của một máy tính thông thường). Cấu trúc danh sách cạnh tuy đơn giản nhưng các phép toán cập nhật đồ thị lại có độ phức tạp tính toán cao. Với những phân tích đó, biểu diễn các cạnh đồ thị có quy mô lớn dựa theo danh sách liền kề là cách tiếp cận phù hợp nhất.

Đối với phương pháp biểu diễn cạnh bằng danh sách liền kề, có hai kỹ thuật để tổ chức danh sách liền kề này:

- **Danh sách các đỉnh cạnh đến của mỗi đỉnh:**
  Với mỗi đỉnh v, tất cả các cạnh đến (incoming edges) đỉnh v trong đồ thị sẽ được lưu trong cùng một danh sách chứa liên tiếp các đỉnh tương ứng ($incoming\_edges$).

- **Danh sách các đỉnh cạnh đi của mỗi đỉnh:**
  Tương tự như phương pháp biểu diễn bằng danh sách các đỉnh cạnh đến của mỗi đỉnh, các cạnh E trong G sẽ được biểu diễn bằng một danh sách các đỉnh liên tiếp đi mỗi đỉnh ($outgoing\_edges$).

Đối với các phép toán trên đồ thị, thông thường có thể quy về các phép toán cơ bản đọc-read và ghi-write dữ liệu. Khi đó, phép ghi dữ liệu sẽ tương ứng với các thao tác thêm hoặc xoá cạnh trong đồ thị; trong khi phép đọc dữ liệu đồ thị tương ứng với các thao tác duyệt, tính khoảng cách tối ưu hay phân tích đồ thị. Mỗi phép toán đọc/ghi dữ liệu đồ thị

nói chung đều có thể được coi như một truy vấn (query) trong đồ thị [111]. Ngoài ra, hầu hết các phép toán đọc trên đồ thị đều liên quan đến việc thi hành phép toán hoặc là duyệt theo chiều sâu trước-DFS, hoặc là duyệt theo chiều rộng trước-BFS [75]. Trong phạm vi luận án này, chúng tôi sẽ tập chung chính vào các phép duyệt đồ thị để tìm đường đi ngắn nhất giữa hai đỉnh. Phép toán này tương ứng với các truy vấn tìm mối quan hệ ngắn nhất để tiếp cận một thành viên trong mạng xã hội, hoặc là nền tảng để phân tích đồ thị.

Với những phân tích đó, các phép toán trên đồ thị hoàn toàn có thể được mô hình hoá dựa theo ba phép toán chính sau:

Định nghĩa 3.1. *Thêm cạnh [$'A'$ u v]: Đây là phép toán hướng đến cập nhật/thay đổi đồ thị hiện thời G bằng việc bổ sung thêm một cạnh khác từ đỉnh đầu u đến đỉnh cuối v. Trong trường hợp cạnh (u, v) đã tồn tại trong tập cạnh E, đồ thị G sẽ không thay đổi. Nếu một trong hai đỉnh (hoặc cả hai) của cạnh mới chưa có trong đồ thị, V phải được cập nhật, bổ sung thêm đỉnh đó.*

Định nghĩa 3.2. *Xoá cạnh [$'D'$ u v]: Phép toán này có nhiệm vụ cập nhật đồ thị hiện thời bằng cách loại bỏ cạnh (u, v) trong tập cạnh E. Trong trường hợp cạnh này không tồn tại trong đồ thị, G phải được đảm bảo không có sự thay đổi.*

Định nghĩa 3.3. *Truy vấn [$'Q'$ u v]: Phép toán này tương ứng với truy vấn tìm khoảng cách của đường đi ngắn nhất từ đỉnh đầu u đến đỉnh cuối v trong đồ thị G. Nếu không tồn tại đường đi giữa chúng hoặc một trong hai đỉnh đó không tồn tại, kết quả trả về là -1. Ngoài ra, khoảng cách ngắn nhất giữa bất kỳ đỉnh nào với chính nó luôn bằng 0.*

Với việc mô hình hoá các thao tác đồ thị dựa trên ba phép toán trên, phần lớn các thao tác trên đồ thị đều có thể được tiến hành thông qua ba phép toán đó.

3.2.2 Bài toán tối ưu hoá truy vấn khoảng cách ngắn nhất trên đồ thị động

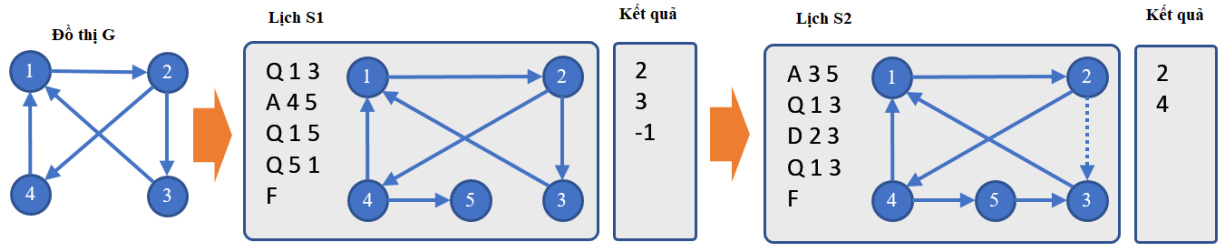
Từ mô hình dữ liệu và thao tác nêu trên, trong luận án này, bài toán nghiên cứu tối ưu hóa truy vấn khoảng cách ngắn nhất trên đồ thị động được phát biểu như sau:
Xét đồ thị $G = (V, E)$ tại thời điểm t có n phép toán tương tranh (concurrent operations) $\{op_1, op_2..., op_n\}$. Mỗi phép toán op_i tương ứng với một trong ba phép toán đã đặc tả ở trên và được biểu diễn bằng bộ ba $op_i = (a_i, u_i, v_i)$; trong đó $a \in [{'A'}, {'D'}, {'Q'}]$ là kiểu phép toán thao tác với cặp đỉnh (u, v).

Yêu cầu tối ưu hóa truy vấn khoảng cách ngắn nhất đặt ra là phải tiến hành xử lý n phép toán đó một cách nhanh nhất có thể mà vẫn đảm bảo được tính nhất quán của G đối với thứ tự thi hành chúng.

Tương tự như việc xử lý với các giao tác tương tranh (concurrent transaction) trong các hệ quản trị CSDL hay các tiến trình tương tranh trong hệ điều hành, tập các phép toán tương tranh trên G sẽ được mô hình hoá quá trình xử lý theo một lịch thi hành S (schedule) các phép toán cập nhật và truy vấn khoảng cách ngắn nhất. Trong S, thứ tự thi hành mỗi thao tác op_i là rất quan trọng, đặc biệt đối với các phép cập nhật G có gây ảnh hưởng đến các phép truy vấn khoảng cách.

Định nghĩa 3.4. *Lịch thi hành $S = \{op_1, op_2..., op_n\}$: Là tập các phép toán $op_i = (a_i, u_i, v_i)$ có thứ tự trên đồ thị G, trong đó $a \in [' A',' D',' Q']$ là kiểu phép toán thao tác với cặp đỉnh (u, v).*

Hình sau đây minh hoạ ví dụ xử lý một số lịch thi hành các phép toán tương tranh trong một đồ thị. Trong ví dụ này, giả sử bản đầu đồ thị G được có 4 đỉnh và 5 cạnh. Tại thời điểm t_1, có tập 4 phép toán đồng thời trên G (với 3 phép truy vấn Q và 1 phép toán thêm cạnh), trong đó thứ tự yêu cầu thực hiện các phép toán đó được thể hiện giống như thứ tự các phép toán trong lịch $S1$. Khi đó, cho dù có áp dụng những phương pháp xử lý song song khác nhau, việc thi hành lịch $S1$ đảm bảo được tính nhất quán nếu [Q 1 3] = 2; [Q 1 5] = 3 và [Q 5 1] = -1. Tương tự như thế đối với lịch $S2$ (với 1 phép thêm cạnh, 1 phép xóa cạnh và 2 phép truy vấn), việc thi hành $S2$ với những phương pháp khác nhau cũng phải đảm bảo kết quả thu được là nhất quán: [Q 1 3] = 2 và [Q 1 3] = 4.



Hình 3.1: Các phép toán tương tranh trên đồ thị

Với mô hình đó, việc giải bài toán đã đặt ra tương ứng với việc tìm lịch thi hành S' tương đương với S để khi thi hành các phép toán trong S' thì thu được cùng tập kết quả truy vấn như khi thi hành S nhưng có thể thi hành một cách tối ưu để nâng cao hiệu năng, đồng thời vẫn phải đảm bảo tính nhất quán của G. Cũng tương tự như khái niệm nhất quán (consistency) trong cơ sở dữ liệu, tính nhất quán của G được định nghĩa như việc đảm bảo dữ liệu đồ thị G chỉ được phép thay đổi theo đúng mong muốn khi thi hành các phép toán đồng thời S. Điều đó có nghĩa việc thi hành các phép toán trong S cho dù có tương tranh luôn phải chuyển đồ thị G từ một trạng thái nhất quán đến một trạng thái nhất quán khác.

Việc thi hành tối ưu S' cần phải được xây dựng dựa theo chiến lược tổ chức dữ liệu đồ thị hợp lý kết hợp với phương pháp khai thác được những mô hình kiến trúc máy tính hiện đại ngày nay (cụ thể như khai thác hết các tài nguyên multi-cores, multi-cpu, dung lượng bộ nhớ cache lớn...).

3.2.3 Cách tiếp cận giải quyết bài toán đặt ra

Từ mô hình bài toán đó, việc thực thi lịch S được chúng tôi tiến hành theo hướng xây dựng giải pháp thi hành trên các hệ thống tính toán đa CPU, nhiều lõi và có không gian bộ nhớ chính lớn. Hiện nay, các hệ thống tính toán có hỗ trợ số luồng (threads) đồng thời từ 30-100 tương đối thông dụng. Hơn nữa, không gian bộ nhớ chính từ 128GB đến 1TB cũng đã rất phổ dụng. Chính vì thế, các giải pháp của chúng tôi sẽ chỉ tập trung nghiên cứu, đề xuất những giải pháp xử lý các lịch thi hành truy vấn tương tranh trên cùng một hệ thống tính toán, chưa cần phải mở rộng thi hành trên các cụm máy tính hay các siêu máy tính.

Giải pháp chính trong việc nâng cao hiệu quả thi hành S hiện nay đều tập trung đến việc song song hoá xử lý các phép toán trong S. Trong chương này, chúng tôi sẽ đề cập chi tiết về các giải pháp được đề xuất trong việc xử lý song song lịch thi hành S này.

3.3 Giải pháp 1: akGroup

Để tiến hành tập các phép toán tương tranh S trên G một cách hiệu quả, giải pháp *akGroup* mà chúng tôi đưa ra dựa trên chiến lược xử lý với ba ý tưởng chính:

1. xây dựng cấu trúc dữ liệu phù hợp để nâng cao tỷ lệ cache hit trong CPU;
2. tối giản không gian tìm kiếm dựa trên việc lựa chọn linh hoạt hàng đợi để duyệt các đỉnh khi tiến hành giải thuật bBFS; và
3. cài đặt hiệu quả cơ chế song song hoá bBFS dựa trên bộ thư viện CilkPlus.

Trong giải pháp này, việc thi hành các phép toán tương tranh trong S được tiến hành

dựa trên cách tiếp cận minh hoạ bằng giải thuật 3.1 sau đây.

Thuật toán 3.1: akGroup: Giải thuật thi hành lịch S

```
1  Function akGroup(G, S)
       Input: Đồ thị G và lịch thi hành các phép toán tương tranh S[a, u, v];
       Output: G được cập nhật và danh sách Dist[.] chứa khoảng cách ngắn nhất theo
               S
2      query_num ← 0 ;
3      Khởi tạo mảng Q ;                    /* lưu truy vấn tính khoảng cách */
4      foreach (a, u, v) ∈ S do
5          if a =′ A′ then
               /* Không cập nhật G nếu (u, v) đã có trong G                  */
6              if v ∈ G.outgoingEdges[u] then continue ;
               /* Thi hành truy vấn trước khi bổ sung cạnh                   */
7              if (query_num > 0) then exec_queries(Q, Dist) ;
8              add_edge(u, v) ;
9          end
10         else if a =′ D′ then
               /* Không cập nhật G nếu (u, v) không tồn tại trong G          */
11             if v ∉ G.outgoingEdges[u] then continue ;
               /* Thi hành truy vấn trước khi xoá cạnh                       */
12             if (query_num > 0) then exec_queries(Q, Dist) ;
13             del_edge(u, v) ;
14         end
15         else if a =′ Q′ then
16             Q[query_num] ← (u, v) ; query_num+ = 1;
17         end
18     end
19     if query_num > 0 then
20         exec_queries(Q, Dist) ;
21     end
22     return Dist[.] ;
```

Trong giải thuật 3.1, các phép toán truy vấn khoảng cách ngắn nhất trong lịch S sẽ được tích lũy lại trong mảng Q. Việc dồn các phép toán truy vấn này cho phép chúng ta có thể tiến hành song song các truy vấn khoảng cách để không ảnh hưởng đến tính nhất quán của đồ thị G.

Đối với các phép toán cập nhật (a, u, v) trong lịch S, chúng sẽ được bắt đầu bằng các phép kiểm tra sự tồn tại của cạnh đó trong G. Nếu cạnh đó đã tồn tại thì không thi hành phép toán thêm cạnh nữa. Và tương tự, nếu cạnh đó không tồn tại trong G, thì sẽ không

thì hành phép xóa cạnh đó nữa. Sau khi kiểm tra xong, trước khi thực hiện phép toán cập nhật đồ thị (thêm/xoá), các truy vấn khoảng cách ngắn nhất sẽ được tiến hành trước, tránh tình trạng phải kiểm tra sau tính đúng đắn của một cạnh khi xét trong các truy vấn liên quan đến cạnh cập nhật. Điều này cũng cho phép toàn bộ các phép toán cập nhật trong S sẽ được tiến hành theo đúng thứ tự xuất hiện và do đó luôn đảm bảo được tính nhất quán của G. Việc thi hành các truy vấn tính khoảng cách sẽ được lưu lại trong danh sách Q và sẽ được thực hiện song song trong $exec\_queries()$.

Như vậy, về bản chất, việc thi hành lịch S trong giải thuật 3.1 được tiến hành theo đúng thứ tự của các phép toán trong S. Vì thế, đồ thị G luôn được đảm bảo tính nhất quán cả trong và sau khi thi hành lịch S.

Phần sau đây sẽ trình bày chi tiết ba ý tưởng chính của giải pháp:

3.3.1 Cấu trúc dữ liệu đồ thị phù hợp

Như đã phân tích ở trên, dữ liệu đồ thị có thể được biểu diễn bằng các danh sách đỉnh liền kề. Tuy nhiên, nếu tổ chức G theo kiểu $|V|$ mảng chứa các đỉnh liền kề đến hay đi của mỗi đỉnh $v \in V$ thì khi cập nhật dữ liệu, vẫn có dữ liệu đồ thị sẽ không thể đảm bảo tính cục bộ (locality, tức được cấp phát trong bộ nhớ chính liên tiếp nhau).

Thông thường, khi thi hành lệnh trong CPU, cả lệnh và/hay dữ liệu sẽ được tải từ không gian bộ nhớ chính nếu như chưa có trong bộ nhớ đệm cache trong CPU. Đó chính là trường hợp "cache miss" và ngược lại, nếu lệnh/dữ liệu đã có trong bộ nhớ cache rồi thì gọi là "cache hit". Bộ nhớ cache của CPU thường được tổ chức theo khối có dung lượng 64B, thông thường được chia làm 2 đến 3 mức cache. Chính vì thế, nếu dữ liệu cần được xử lý liên tiếp mà được tổ chức có tính cục bộ thì sẽ tăng được tỷ lệ cache hit khi thi hành. Sự khác biệt về thời gian khi xử lý tương ứng với các trường hợp cache miss/cache hit được minh hoạ như thống kê trong bảng 3.1[2]:

Bảng 3.1: Thời gian thao tác bộ nhớ

	Chi phí (ns)	**Chú giải**
Tham chiếu mức cache L1	0.5	> 2 ALU instruction latency
Dự đoán sai rẽ nhánh	3	
Tham chiếu mức cache L2	4	8x L1 cache
Mutex lock/unlock	17	
Tham chiếu bộ nhớ chính	100	25x L2 cache, 200x L1 cache

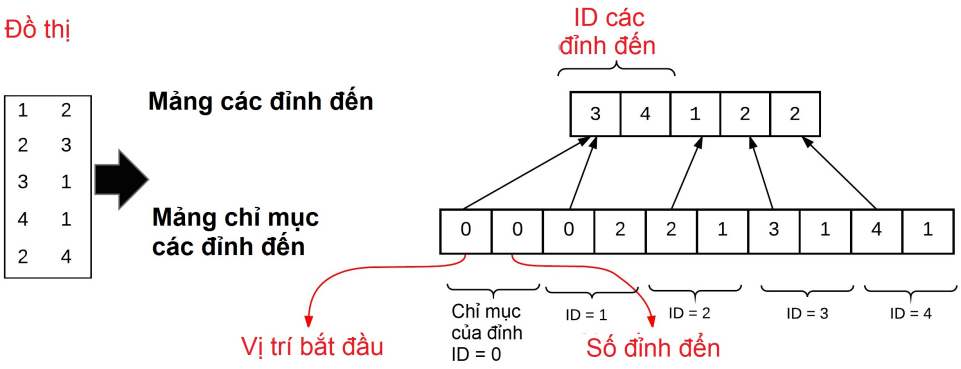
Với những phân tích đó, dữ liệu đồ thị cần phải được lưu trữ liên tiếp nhau trong không gian bộ nhớ. Khi đó, mỗi khi truy cập một đỉnh v, các đỉnh cùng mức với v sẽ có xác suất

[2]Thông tin thêm có thể đọc tại `http://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html`

đã được đưa vào trong bộ nhớ cache cao hơn. Giảm được việc truy xuất đến bộ nhớ chính rõ ràng sẽ nâng cao được hiệu quả trong xử lý các truy vấn đồ thị.

Từ ý tưởng đó, dữ liệu đồ thị G sẽ được biểu diễn như sau:

- Mỗi đỉnh v sẽ được định danh bằng một số tự nhiên nằm trong khoảng $[0..(|V|-1)]$
- Các đỉnh đến/đi của một đỉnh sẽ được lưu trữ trong một vùng liền kề của một mảng tương ứng gọi là incoming_edges/outgoing_edges.
- Việc định vị và số lượng các đỉnh liền kề của v trong mảng incoming_edges / outgoing_edges đó sẽ được tiến hành dựa theo mảng chỉ mục incoming_index / outgoing_index. Để tăng tỷ lệ "cache hit", số đỉnh liền kề num và vị trí bắt đầu pos trong $incoming\_edges$ / $outgoing\_index$ sẽ được lưu theo cặp kề nhau (pos, num).



Hình 3.2: Minh hoạ cấu trúc dữ liệu đồ thị

Để phục vụ cho việc duyệt đồ thị theo cả hai chiều, chúng tôi sẽ tiến hành biểu diễn G với cả hai mảng các đỉnh liền kề đến và liền kề đi. Với phương pháp đó, đồ thị G sẽ có cấu trúc dữ liệu gồm:

- $incoming\_edges, incoming\_index$ kiểu Integer (4-bytes) để biểu diễn tất cả các cạnh theo kiểu đỉnh liền kề đến trong G, trong đó $incoming\_index$ để lưu cặp vị trí và số lượng đỉnh liền kề (pos, num).
- $outgoing\_edges, outgoing\_index$ cũng có kiểu Integer để biểu diễn tất cả các cạnh theo kiểu liền kề đi trong G. Tương tự, các cặp (pos, num) được lưu trong mảng chỉ mục $outgoing\_index$ để xác định vị trí bắt đầu và số đỉnh đi của mỗi đỉnh trong G.

Việc sử dụng cả danh sách liền kề đến và đi cho phép duyệt đồ thị theo cả hai hướng đối với các truy vấn tìm đường đi theo giải thuật bBFS[19].

3.3.2 Tối ưu hoá các phép toán cập nhật

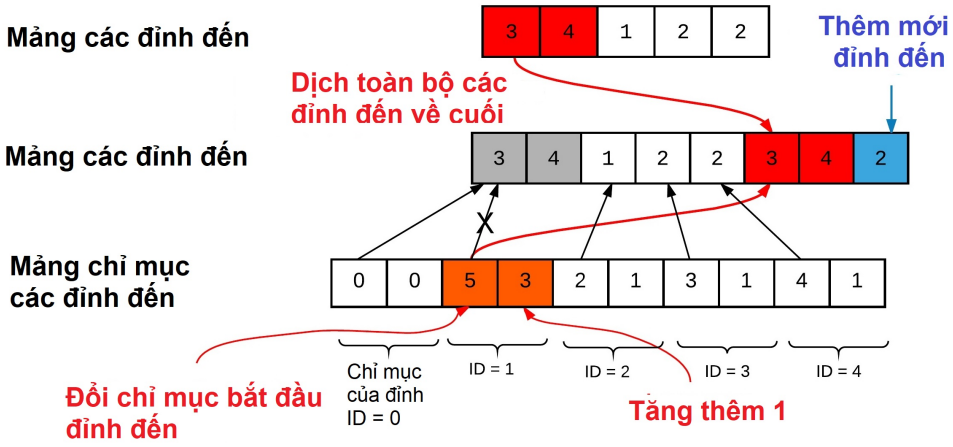
Việc cập nhật đồ thị G với cấu trúc dữ liệu nêu trên sẽ được tiến hành qua các thủ tục thêm và xoá cạnh trên G.

3.3.2.1 Thêm cạnh mới

Dựa vào cấu trúc dữ liệu nêu trên, chúng tôi xem xét hai ý tưởng chính để thực hiện việc thêm cạnh mới như sau:

- *Cấp phát trước bộ nhớ "bucket"*
  Trong mảng chứa đỉnh liền kề, sau mỗi 32 đỉnh sẽ có dự phòng trước một vùng nhớ để phục vụ thêm các đỉnh đến/đi của cạnh mới. Vùng nhớ này được gọi tắt là BUCKET. Giá trị mặc định của BUCKET là 8, có nghĩa là tối đa 8 đỉnh có thể được bổ sung vào sau mỗi đoạn 32 đỉnh trong mảng dữ liệu đỉnh liền kề.

- *Dịch về cuối danh sách*
  Dịch toàn bộ các cạnh đến/đi liên quan đến hai đỉnh của cạnh thêm mới về cuối danh sách chứa các đỉnh đến/đi *incoming_edges*/*outgoing_edges* như minh hoạ ở hình 3.3.



Hình 3.3: Phép toán bổ sung thêm cạnh trên đồ thị

Dựa trên các thử nghiệm thực tế, chúng tôi thấy ý tưởng thứ hai cho thời gian thi hành hiệu quả hơn so với ý tưởng thứ nhất. Lý do nằm chính trong cách tổ chức dữ liệu của chúng: ý tưởng thứ hai cho phép khởi tạo được mảng chứa các đỉnh liên tiếp nhau, trong khi với ý tưởng thứ nhất, chúng ta lại tạo ra nhiều khe trống (slots) trong danh sách đỉnh liền kề. Từ đó có thể dẫn đến tỷ lệ "cache miss" cao hơn so với ý tưởng thứ hai.

Như đã trình bày ở đầu mục, để đảm bảo được G luôn nhất quán trong quá trình thi hành các thao tác trong S, các truy vấn khoảng cách cần phải được thi hành trước. Từ đó,

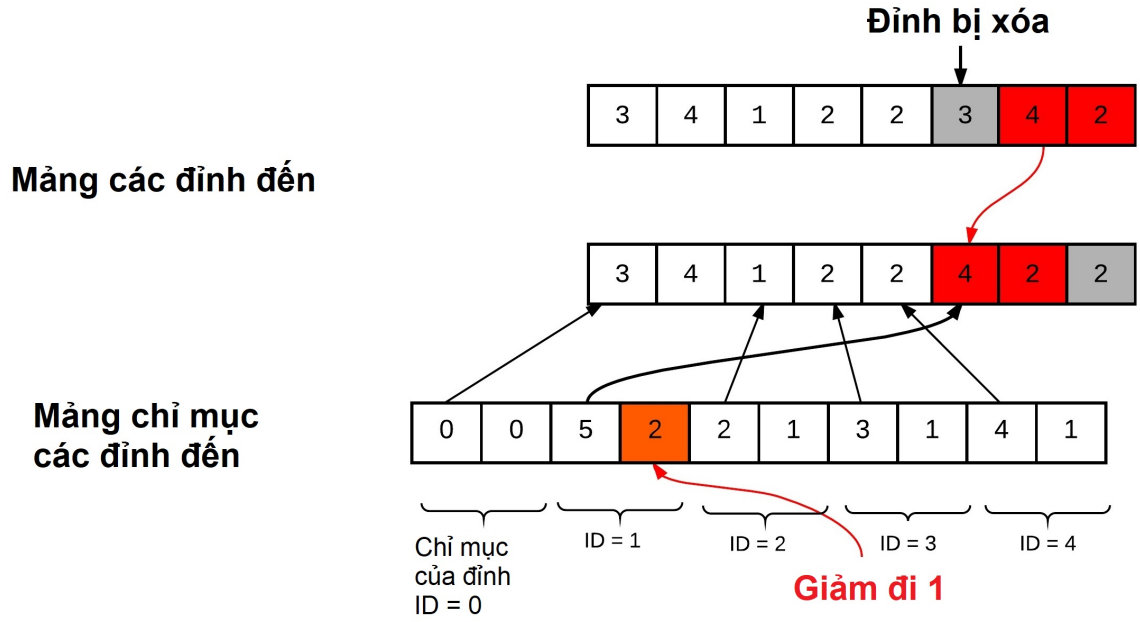
thủ tục thêm cạnh $add\_edge(u, v)$ vào đồ thị G được minh hoạ như giải thuật 3.2:

Thuật toán 3.2: akGroup: Thêm cạnh (u, v) vào đồ thị G

1 **Function** $add\_edge(u, v)$
  Input: G được biểu diễn bởi
    $incoming\_edges, incoming\_index, outgoing\_edges, outgoing\_index$
  Output: G được cập nhật cạnh (u, v)
  `/* Cập nhật` u `vào mảng liền kề đến` $incoming\_edges$ `*/`
2 $incoming\_index[v].num+ = 1$;
3 $incoming\_index[v].pos = incoming\_edges.length$;
4 Chuyển các đỉnh đến của v về cuối $incoming\_edges$ và thêm u;
  `/* Cập nhật` v `vào mảng liền kề đi` $outgoing\_edges$ `*/`
5 $outgoing\_index[u].num+ = 1$;
6 $outgoing\_index[u].pos = outgoing\_edges.length$;
7 Chuyển các đỉnh đi từ u về cuối $outgoing\_edges$ và thêm v ;

3.3.2.2 Xoá một cạnh

Với cấu trúc dữ liệu đồ thị như đã mô tả, phép toán xoá cạnh được thực hiện đơn giản hơn nhiều so với phép toán thêm cạnh. Thao tác này chỉ cần thực hiện việc giảm số đỉnh đến/đi trong mảng chỉ mục và loại đi đỉnh tương ứng của cạnh cần xoá trong danh sách đỉnh đến/đi incoming/outgoing_edges. Phép toán này được minh hoạ như hình 3.4 dưới đây:



Hình 3.4: Thi hành phép toán xoá cạnh trong đồ thị

Cũng tương tự như phép toán thêm cạnh, để đảm bảo tính nhất quán và kết quả truy

vấn trả về chính xác, nếu việc xoá cạnh liên quan đến các truy vấn tính khoảng cách ngắn nhất, toàn bộ các truy vấn lưu trong Q sẽ được thi hành trước khi tiến hành xoá cạnh. Từ đó, thao tác xoá cạnh được minh hoạ theo thuật toán 3.3 sau:

Thuật toán 3.3: akGroup: Xoá cạnh (u, v) trong đồ thị G

```
1 Function del_edge(u,v)
    Input: G được biểu diễn bởi
           incoming_edges, incoming_index, outgoing_edges, outgoing_index; Q
           chứa danh sách các truy vấn khoảng cách hiện thời
    Output: G không còn cạnh (u, v)
    /* Cập nhật thông tin các đỉnh đến                                  */
2   Loại đỉnh u trong danh sách các đỉnh đến của v trong incoming_edges ;
3   incoming_index.num− = 1 ;
    /* Cập nhật thông tin các đỉnh đi                                   */
4   Loại đỉnh v trong danh sách các đỉnh đi của u trong outgoing_edges ;
5   outgoing_index.num− = 1 ;
```

3.3.3 Tối ưu các truy vấn

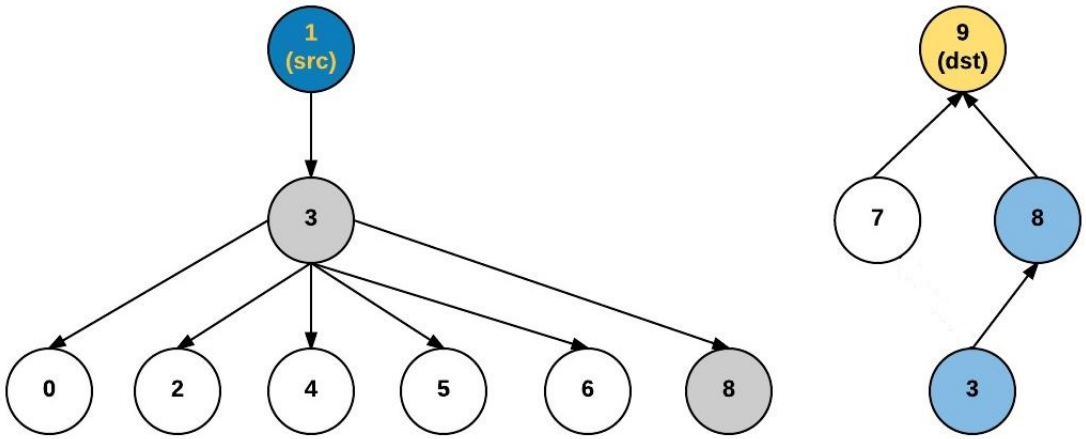
Trong giải pháp này, các truy vấn tính khoảng cách ngắn nhất giữa hai đỉnh sẽ được tích luỹ vào trong danh sách Q để tiến hành song song trong hệ thống tính toán. Như đã phân tích ở chương 2, BFS là giải thuật hiệu quả nhất để tính khoảng cách ngắn nhất giữa hai đỉnh trong đồ thị không trọng số, có hướng và có quy mô lớn. Tuy nhiên, với truy vấn tính khoảng cách ngắn nhất từ u đến v, để nâng cao hiệu quả thi hành BFS, cần phải tiến hành theo cả hai chiều đi từ u và lần ngược từ đỉnh đến v (giải thuật bidirectionalBFS-bBFS) kết hợp với kỹ thuật sử dụng 1 bit để lưu vết đã duyệt trong hàng đợi [19][58]. Việc thi hành song song Q sẽ được tiến hành theo các kỹ thuật tính toán hiệu năng cao nhằm khai thác hết năng lực tính toán của các hệ thống tính toán hiện đại multicore, multi-CPU. Từ đó, việc thi hành các truy vấn trong Q sẽ dựa vào giải thuật tính đường đi ngắn nhất $ComputeShortest1(u, v)$ và phương pháp song song hoá các truy vấn trong Q.

3.3.3.1 Giải thuật tính khoảng cách ngắn nhất

Việc tính khoảng cách ngắn nhất từ u đến v trong đồ thị có hướng không trọng số G sẽ dựa trên giải thuật bBFS. Ý tưởng chính của bBFS là duyệt BFS theo cả hai chiều: từ u duyệt dần theo chiều rộng trước và từ v lần ngược lên cũng theo chiều rộng. Quá trình duyệt chỉ kết thúc khi tồn tại đỉnh nằm trong cả hàng đợi duyệt của chiều đi và chiều đến. Khi đó, khoảng cách ngắn nhất chính là tổng khoảng cách từ u đến đỉnh nằm chung đó và từ đỉnh này đến v.

Để giảm kích thước các hàng đợi duyệt BFS, chúng ta sẽ sử dụng một bit để xác lập trạng

thái một đỉnh là đã duyệt hay chưa được duyệt trong G. Ngoài ra, việc lựa chọn chiều duyệt trong bBFS cũng sẽ ảnh hưởng đến hiệu năng thi hành của giải thuật $ComputeShortest1$. Quả vậy, xét trường hợp duyệt BFS theo hai chiều như minh hoạ ở hình 3.5, nếu chúng ta chọn chiều duyệt dựa vào số lượng đỉnh trong mỗi hàng đợi đi/đến như công trình [19] và với truy vấn tìm đường đi ngắn nhất từ đỉnh nguồn 1(src) đến đỉnh đích 9(dst). Khi đó, chúng ta chọn hàng đợi đi của đỉnh nguồn 1 trước (vì chỉ có 1 đỉnh con) và đỉnh đi 3 sẽ được duyệt trước. Khi đó, sẽ có (1+6=7) đỉnh được đưa vào hàng đợi đi và đỉnh đưa vào hàng đợi đến (khi xét đỉnh đích 9).



Hình 3.5: Duyệt hai chiều BFS để tính khoảng cách ngắn nhất

Để giảm không gian tìm kiếm (thông qua giảm số đỉnh phải duyệt), chúng tôi đã đề xuất ý tưởng tinh chỉnh giải thuật bBFS với chiến lược phân tích ở mức sâu hơn như sau: khi xét mỗi đỉnh (mức 0) chúng ta sẽ chọn chiều duyệt tiếp dựa trên hàng đợi chiều nào có tổng các đỉnh con (mức 1) và các đỉnh cháu (mức 2) bé nhất. Chẳng hạn, với trường hợp như minh hoạ ở hình 3.5, khi duyệt đỉnh 1, hàng đợi đi sẽ có 1 đỉnh con mức 1 và 6 đỉnh cháu mức 2, tổng là 7. Trong khi đó, nếu xét đỉnh 9 trước, hàng đợi đến sẽ có 2 đỉnh con mức 1 và 1 đỉnh cháu mức 2, tổng là 3. Như vậy có thể thấy nếu chúng ta chọn chiều đến để duyệt đỉnh 9 trước, chỉ cần 2 lần duyệt là chúng ta đã tìm được đường đi ngắn nhất; trong khi nếu chúng ta chọn duyệt theo chiều đi từ đỉnh 1, chúng ta phải duyệt qua 7+2=9 đỉnh mới tìm được đường đi ngắn nhất. Từ đó, có thể thấy việc tính toán chọn chiều duyệt dựa trên chiến thuật đã nêu hiệu quả hơn.

Tóm lại, chiến thuật duyệt bBFS của chúng tôi đối với các truy vấn tính khoảng cách ngắn nhất được mô tả như sau:

- Sử dụng hai mảng bitmaps đi và đến để mỗi bit xác lập trạng thái đỉnh tương ứng vị trí bit đó đã được duyệt hay chưa.
- Sử dụng chiến lược lựa chọn chiều duyệt BFS có ghi nhận thêm số lượng các cháu trong hàng đợi: theo chiều đi hay đến dựa trên tổng số đỉnh con và đỉnh cháu ở mỗi

hàng đợi.

Từ đó, các truy vấn tính khoảng cách ngắn nhất trên G sẽ được thực hiện dựa trên sự kết hợp giải thuật 2.3 ở chương 2 với kỹ thuật sử dụng mảng bitmaps lưu trạng thái (nhằm giảm kích thước hàng đợi và tăng tỷ lệ cache hit) và chiến lược duyệt bBFS nêu trên. Đồ thị G sẽ được biểu diễn bởi các mảng liền kề $incoming\_edges$, $incoming\_index$, $outgoing\_edges$, $outgoing\_index$; cho phép duyệt được theo cả hai chiều. Hai mảng bitmap $incoming\_map$ và $outgoing\_map$ được cấp phát trước để phục vụ lưu vết các đỉnh đã duyệt. Quá trình thi hành các bước chính trong giải thuật tính khoảng cách ngắn nhất 3.4 được minh hoạ chi

tiết trong hàm *ComputeShortest*1 sau đây.

Thuật toán 3.4: akGroup: Giải thuật tính khoảng cách ngắn nhất (u, v)

```
1  Function ComputeShortest1(u, v)
       Input: G được biểu diễn bởi incoming_edges, incoming_index, outgoing_edges, outgoing_index;
              incoming_map, outgoing_map là hai mảng bitmaps để đánh dấu các đỉnh đã duyệt
       Output: Giá trị là khoảng cách ngắn nhất từ u đến v
2      if u == v then return 0 ;
3      if (incoming_num[v] == 0)or(outgoing_num[u] == 0) then return-1 ;
4      in_queue ← v; set_bit(v, in_map) ;                    /* khởi tạo in_queue và đánh dấu đã duyệt v */
5      out_queue ← u; set_bit(u, out_map ;                   /* khởi tạo out_queue và đánh dấu đã duyệt u */
6      in_cost = 0; out_cost = 0 ;                           /* khoảng cách tới/từ u/v */
7      count = 1; out_size = outgoing_num[u]; in_size = incoming_num[v];
8      while (count > 0) do
9          if (out_size < in_size) then                      /* theo hướng đi từ đỉnh nguồn */
10             out_size = 0; out_cost+ = 1; count = 0 ;
11             while out_queue not empty do
12                 e ← out_queue ;
13                 foreach n ∈ outgoing_edges[e] do
14                     if test_bit(n, out_map) then
15                         if n == v then                    /* n là đỉnh đích */
16                             Clear all bits of in/out maps; return out_cost ;
17                         end
18                         if test_bit(n, in_map) then Clear all bits of in/out maps; return in_cost + out_cost ;
19                         out_queue ← n; count+ = 1; out_size+ = outgoing_num[n]; set_bit(n, out_map);
20                     end
21                 end
22             end
23             out_size+ = count;
24         end
25         else                                              /* theo hướng ngược từ đỉnh đích */
26             in_size = 0; in_cost+ = 1; count = 0;
27             while in_queue not empty do
28                 e ← in_queue ;
29                 foreach n ∈ incoming_edges[e] do
30                     if test_bit(n, in_map) then
31                         if n == u then                    /* n là đỉnh nguồn */
32                             Clear all bits of in/out maps; return in_cost ;
33                         end
34                         if test_bit(n, out_map) then
35                             Clear all bits of in/out maps; return in_cost + out_cost
36                         end
37                         in_queue ← n; count+ = 1; in_size+ = incoming_num[n]; set_bit(n, in_map) ;
38                     end
39                 end
40             end
41             out_size+ = count;
42         end
43     end
44     Clear all bits of in/out maps ;
45     return-1 ;
```

3.3.3.2 Xử lý song song truy vấn

Để có thể tiến hành song song các truy vấn tìm khoảng cách ngắn nhất, giải pháp của chúng tôi đề xuất dựa vào các ý tưởng sau:

- Song song hoá quá trình thực hiện các truy vấn tính khoảng cách trên các luồng khác nhau chứ không song song quá trình tính khoảng cách ngắn nhất. Cách tiếp cận này cho phép trong mỗi luồng, việc tính toán trên dữ liệu đồ thị khai thác được tính cục bộ dữ liệu đồ thị, từ đó nâng cao được hiệu năng bộ nhớ đệm cache.

- Sử dụng hai hàng đợi toàn cục để chứa các đỉnh đến/đi khi duyệt đồ thị (gọi là *incoming_queue* và *outgoing_queue*); Việc xử lý truy vấn tìm khoảng cách ngắn nhất được tiến hành trong một luồng riêng biệt và sử dụng một mảng *incoming_queue* và *outgoing_queue* riêng cho luồng đó.

- Hai mảng toàn cục để lưu vết các đỉnh đã duyệt (*in_maps* và *out_maps*). Hai mảng lưu vết này sẽ được cấp phát trước và mỗi mảng có kích thước là $\frac{Max\_V}{32} * threadNum$ phần tử kiểu Integer (4-bytes) với $threadNum$ là số luồng song song, Max_V tương ứng với số đỉnh lớn nhất có thể có trong đồ thị G. Việc sử dụng bitmap để đánh dấu trạng thái duyệt đỉnh cho phép giảm kính thước đi 32 lần so với việc sử dụng một từ kiểu Integer. Ngoài ra, việc cấp phát trước hai mảng này tương ứng với số luồng song song sẽ cho phép chúng ta giảm thời gian khởi tạo hai mảng này. Để tránh việc phải mất nhiều thời gian khởi tạo các mảng này khi kích thước $|V|$ lớn, *in_maps* và *out_maps* sẽ phải được xoá hết trạng thái đã bật khi kết thúc để có thể sử dụng lại trong lần truy vấn kế tiếp trong luồng đó.

- Giải pháp song song sẽ được cài đặt dựa trên bộ thư viện CilkPlus do Intel xây dựng [49]. Việc lựa chọn CilkPlus cũng đã được chúng tôi phân tích bằng cả thực nghiệm lẫn tham khảo các công trình liên quan [59]. Ngoài ra, chúng tôi cũng đã tiến hành các thử nghiệm đánh giá với cả CilkPlus, OpenMP và pThread, các kết quả thử nghiệm cũng đã minh chứng hiệu năng của CilkPlus là tốt hơn so với cả hai bộ thư viện còn lại.

Từ đó, hàm *exec_queries* sẽ được tiến hành theo giải thuật sau:

Thuật toán 3.5: akGroup: Thi hành song song các truy vấn tính khoảng cách ngắn nhất trong G

```
1 Function exec_queries(Q, Dist)
      Input: Đồ thị G và tập các truy vấn Q
      Output: Danh sách Dist chứa giá trị khoảng cách ngắn nhất tương ứng với các
              truy vấn trong S
      /* Thi hành song song vòng For sử dụng thư viện CilkPlus          */
2     for i = 0 to query_num do
3         (u, v) ← Q[i] ;
4         Dist[i] = ComputeShortest1(u, v);
5     end
6     return Dist[.] ;
```

3.3.4 Đánh giá thuật toán

Về mặt lý thuyết, việc thi hành lịch các phép toán đồng thời S theo phương pháp trước khi thực hiện các phép toán cập nhật thì tiến hành song song các truy vấn khoảng cách ngắn nhất luôn đảm bảo được tính đúng đắn của các phép toán trên đồ thị. Rõ ràng lý do nằm ở các phép toán cập nhật được thực hiện tuần tự nên luôn đảm bảo tính nhất quán của đồ thị. Trong khi các truy vấn khoảng cách ngắn nhất, là các truy vấn thuộc dạng chỉ đọc dữ liệu, luôn được thực hiện trên dữ liệu đồ thị nhất quán nên luôn đảm bảo được tính đúng đắn của việc thi hành lịch S.

Tuy nhiên, việc tổ chức dữ liệu như đã trình bày trong giải pháp nêu trên nhìn chung chưa mang lại hiệu quả cao trong việc thi hành các phép toán cập nhật. Hơn nữa, việc xử lý di chuyển toàn bộ các đỉnh liền kề về cuối mảng khi thi hành nhiều lần lại tạo ra những khe hổng dữ liệu, làm giảm tỉ lệ cache hit. Đây cũng là lý do khi chúng tôi mang giải pháp này tham gia cuộc thi ACM SigMod Programming Contest năm 2016 thì chỉ đạt giải Ba [97].

Đối với giải thuật tìm đường đi ngắn nhất bBFS có sử dụng mảng bitmaps để đánh dấu đỉnh đã duyệt và bổ sung thêm chiến lược lựa chọn dựa vào tổng số con và cháu của mỗi đỉnh đã nâng cao hiệu năng của giải thuật khi thi hành. Tuy nhiên, độ phức tạp tính toán của giải thuật $ComputeShortest1$ vẫn là $O(|V| + |E|)$ với trường hợp tồi nhất là phải duyệt tất cả đỉnh và cạnh của G mới tìm được đường đi ngắn nhất. Trong thực tế, nếu giả sử khoảng cách ngắn nhất giữa hai đỉnh (u, v) là k và trung bình mỗi đỉnh $v \in G$ có b cạnh đi/đến, khi đó nếu tiến hành giải thuật BFS thông thường, chúng ta sẽ phải duyệt trung bình qua số đỉnh là $1 + b + b^2 + ... + b^k$, tức có độ phức tạp là $O(b^k)$. Trong khi sử dụng bBFS duyệt cả hai chiều, số đỉnh cần duyệt trung bình sẽ là $2 + 2b + 2b^2 + ... + 2b^{\frac{k}{2}}$, tức độ phức tạp lúc bấy giờ chỉ còn $O(b^{\frac{k}{2}})$. Như vậy, trong trường hợp bình thường, bBFS có độ phức tạp giảm một nửa hàm mũ so với BFS.

Việc đánh giá thông qua thực nghiệm sẽ được chúng tôi trình bày chi tiết ở mục 3.6 của chương này.

3.4 Giải pháp 2: akGroupPlus

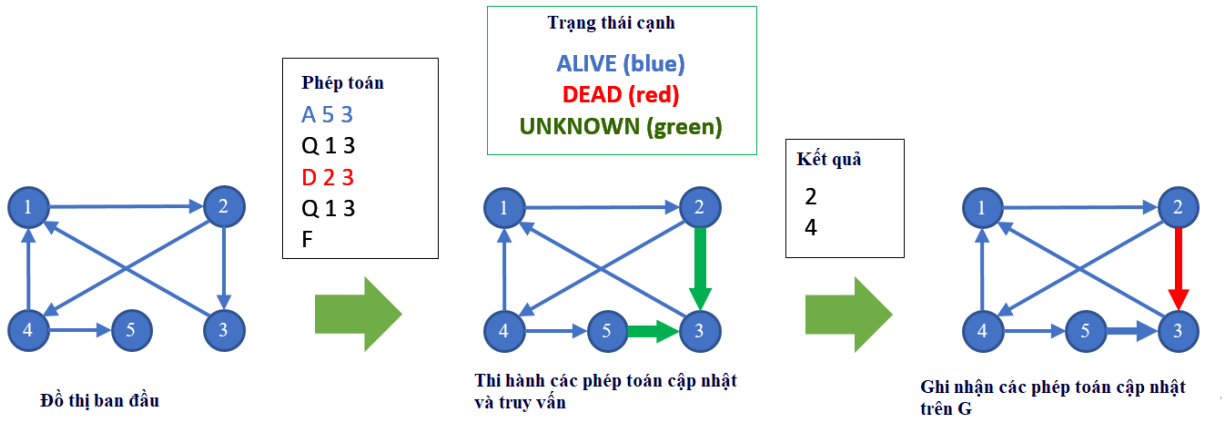
Như đã đánh giá sơ lược ở trên, giải pháp 1 khi xử lý các phép toán tương tranh trong lịch thi hành S trên đồ thị G chưa thực sự hiệu quả ở các phép toán cập nhật. Đây chính là động lực chính để chúng tôi tiếp tục nghiên cứu, đề xuất những cải tiến trong việc mô hình hoá dữ liệu đồ thị và các phép toán tương tranh đối với bài toán đã đặt ra trong luận án.

Việc cải tiến giải pháp 1 cũng sẽ được chúng tôi thực hiện trên cả hai phương diện: thay đổi cách thức tổ chức dữ liệu cho phù hợp hơn và áp dụng những kỹ thuật song song hoá các truy vấn một cách hiệu quả hơn đối với cả các phép toán cập nhật lẫn truy vấn tính khoảng cách ngắn nhất.

3.4.1 Tổ chức dữ liệu đồ thị kèm trạng thái

Trong giải pháp 2 này, dữ liệu đồ thị sẽ được tổ chức dưới dạng mảng danh sách các đỉnh liền kề. Chúng tôi sẽ không tổ chức dồn tất cả các mảng liền kề thành một mảng như trong giải pháp 1, thay vào đó chúng tôi sử dụng mảng $incomingEdges/outgoingEdges$ có $|V|$ phần tử mà mỗi phần tử chứa một danh sách đỉnh liền kề đến hoặc đi.

Để có thể song song hoá nhiều truy vấn đồng thời, chúng tôi sử dụng giải pháp lưu lại vết các cạnh được cập nhật trong quá trình thi hành lịch S. Đây là ý tưởng đã được nhóm H_minor_free[46] sử dụng, với việc mô hình hoá các cạnh bởi 3 trạng thái: ALIVE tương ứng với cạnh đó tồn tại trong G; DEAD để thể hiện cạnh đó đã bị xoá khỏi G và UNKNOWN để thể hiện cạnh đó đang được cập nhật bởi các phép toán trong S. Như vậy, các phép toán cập nhật cạnh sẽ thiết lập trạng thái UNKNOWN cho các cạnh đó. Khi xử lý các truy vấn tính khoảng cách ngắn nhất, nếu gặp cạnh có trạng thái UNKNOWN, ta cần phải xét kỹ hơn thứ tự thi hành các phép toán trong lịch S để quyết định cạnh này có được sử dụng hay không. Kết thúc việc thi hành các phép toán trong S chúng ta mới tiến hành ghi nhận chính thức các cạnh có trạng thái UNKNOWN về thành ALIVE hay DEAD tuỳ thuộc vào phép tính cập nhật cuối cùng liên quan đến cạnh đó. Việc sử dụng trạng thái các cạnh khi thi hành các phép toán trong S có thể được minh hoạ như hình 3.6 sau đây:



Hình 3.6: Thi hành các phép toán tương tranh có thể hiện trạng thái

Nếu chúng ta thu hẹp phạm vi chỉ xét đến những đồ thị có số đỉnh $|V| < 2^{30} = 1.073.741.824$ (tức chưa xét đến những đồ thị kiểu như Facebook chẳng hạn), khi đó, định danh mỗi đỉnh chỉ cần dùng 30 bits thay vì đủ 4-bytes như giải pháp 1. Khi đó, các đỉnh trong danh sách liền kề đến/đi của mỗi đỉnh có thể dành 2 bits cuối để mã hoá trạng thái của cạnh đó (vì chỉ có 3 trạng thái).

Từ ý tưởng đó, dữ liệu đồ thị cũng sẽ được tổ chức theo cả chiều đi lẫn chiều đến như sau:

- (i) Mỗi đỉnh được định danh bằng một số 4-bytes;

- (ii) Các đỉnh cạnh đến/đi của đỉnh u sẽ được lưu trong vector có sắp xếp thứ tự tăng dần $incomingEdges[u]/outgoingEdges[u]$. Mỗi phần tử v kích thước 4 bytes của vector đó sẽ được mã hoá như sau: 30 bits trái nhất đầu tiên để định danh đỉnh đến/đi; 2 bits còn lại để thể hiện trạng thái của cạnh (u, v) (ALIVE, DEAD, UNKNOWN có thể được gán tương ứng là 00, 01, 10 chẳng hạn). Ban đầu, với mỗi $(u, v) \in E$, giá trị các đỉnh u/v được lưu trong $incomingEdges[v]/outgoingEdges[u]$ với trạng thái 2 bits cuối tương ứng với ALIVE.

Việc tổ chức các danh sách đỉnh liền kề theo cả chiều đi lẫn chiều đến sẽ cho phép chúng ta duyệt BFS theo cả hai chiều[19].

3.4.2 Xử lý các phép toán tương tranh

Dựa vào phương pháp tổ chức dữ liệu đồ thị nêu trên, việc thi hành các phép toán tương tranh trong S sẽ được xử lý theo các bước sau:

- Lưu lại có thứ tự các phép toán cập nhật vào mảng Updates; các phép toán truy vấn đường đi ngắn nhất vào mảng Queries;
- Tiến hành xử lý tuần tự các phép toán cập nhật và đánh dấu các cạnh được cập nhật về trạng thái UNKNOWN;
- Tiến hành xử lý song song các truy vấn đường đi ngắn nhất;
- Cập nhật chính thức trạng thái các cạnh liên quan đến các phép toán cập nhật, tức chuyển trạng thái các cạnh đó từ UNKNOWN về DEAD (nếu bị xoá) hay ALIVE (nếu được thêm vào);

Các bước xử lý trên có thể được minh hoạ cụ thể hơn qua giải thuật 3.6 dưới đây:

Thuật toán 3.6: akGroupPlus: Giải thuật thi hành lịch S

```
1  Function akGroupPlus(G, S)
       Input: Đồ thị G và danh sách S có n phép toán (a, u, v) với 'a' là phép toán
       Output: G ghi nhận tất cả cập nhật trong S và danh sách khoảng cách ngắn
               nhất Dist[.] của tất cả truy vấn 'Q'
2      for t=0; t < n; t++ do
3          (a,u,v) = S[t];
4          if a ='Q' then
5              Queries.push_back(t,u,v) ;      /* đẩy bộ (t,u,v) vào cuối Queries */
6          end
7          else
8              Updates.push_back(t,a,u,v) ;   /* đẩy (t,a,u,v) vào cuối Updates  */
9          end
10     end
11     UpdateSerial(Updates) ;     /* Cập nhật trạng thái các cạnh theo giải
           thuật 3.7  */
12     Dist[] = ParallelQuery(Queries) ;   /* Thi hành song song các truy vấn
           tính khoảng cách ngắn nhất 'Q' theo giải thuật 3.11  */
13     CommitSerial(Updates) ;  /* Ghi nhận các cạnh thêm/xoá trong G theo
           giải thuật 3.8  */
14     return Dist[.] ;
```

Trong giải thuật này, mỗi bộ khi đưa vào cả hai vectors *Queries* và *Updates* đều kèm theo một tham số nhãn thời gian t để xác định thứ tự phép toán trong lịch S. Tham số này sẽ được sử dụng trong các giải thuật cập nhật và tính khoảng cách để xác định trạng thái chính xác mỗi cạnh UNKNOWN tại thời điểm xử lý mỗi truy vấn tính khoảng cách.

Về tính đúng đắn của giải thuật 3.6, có thể thấy tất cả các đỉnh cập nhật liên quan đến phép toán truy vấn khoảng cách ngắn nhất đều được kiểm tra, xác định liệu có thể xét đỉnh đó trong phép toán đó hay không. Vì thế, tất cả các phép toán trong S luôn duy trì được tính nhất quán của đồ thị G và kết quả các phép toán truy vấn khoảng cách trả về là hoàn toàn giống như thi hành lịch S một cách tuần tự.

3.4.3 Tối ưu hoá các phép toán cập nhật

Tất cả các phép toán cập nhật (thêm/xoá) cạnh trong vector *Updates* sẽ được thi hành bằng cách cập nhật trước tiên trạng thái UNKNOWN đối với các đỉnh liên quan đến các cạnh thêm/xoá trong *incomingEdges*/*outgoingEdges*. Việc ghi nhận chính thức các phép toán cập nhật sẽ được tiến hành sau khi đã hoàn thành quá trình xử lý các truy vấn khoảng cách ngắn nhất trong *Queries*. Trong quá trình xử lý các truy vấn đó, nhãn thời gian sẽ

được sử dụng để xác định khi nào thì ghi nhận các cạnh có trạng thái UNKNOWN.

Thuật toán 3.7: akGroupPlus: Cập nhật các cạnh trong lịch S

```
1 Function UpdateSerial(Updates)
    Input: Updates: vector chứa các phép toán cập nhật kèm nhãn thời gian
           (t, a, u, v); đồ thị G;
    Output: G được cập nhật trạng thái một số cạnh bởi Updates
2   foreach (t, a, u, v) ∈ Updates do
3     if a == 'A' then
4       InsertNode(outgoingEdges[u], v); InsertNode(incomingEdges[v], u) ;
5     end
6     else
7       RemoveNode(outgoingEdges[u], v); RemoveNode(incomingEdges[v], u) ;
8     end
9   end
```

Trong giải thuật này, hàm $InsertNode(sortedVector, v)$ có chức năng bổ sung đỉnh v vào đúng vị trí tương ứng với giá trị v trong vector $sortedVector$ được sắp xếp theo thứ tự đỉnh tăng dần. Việc sử dụng vector có thứ tự này cho phép tìm nhanh được vị trí cần bổ sung, hoặc phải loại bỏ (sử dụng tìm kiếm nhị phân). Nếu v đã có trong $sortedVector$ và có trạng thái ALIVE, InsertNode sẽ không thay đổi vector $sortedVector$. Nếu không, v sẽ được thiết lập trạng thái UNKNOWN để trong cả hai trường hợp v chưa có trong $sortedVector$ và v có nhưng trạng thái không phải ALIVE. Việc xác định trạng thái UNKNOWN để cho phép chúng ta đánh dấu cần phải xét đỉnh v kỹ hơn (cụ thể là dựa vào thứ tự trong lịch thi hành) để có ghi nhận hay không khi thi hành các phép toán.

Tương tự, hàm $DeleteNode(sortedVector, v)$ đảm nhiệm việc cập nhật trạng thái loại bỏ một cạnh tương ứng với đỉnh v trong $sortedVector$. Nếu đỉnh v không tồn tại trong $sortedVector$ hoặc có tồn tại nhưng trạng thái của v là DEAD, $sortedVector$ sẽ không đổi. Ngược lại, đỉnh v sẽ được gán trạng thái UNKNOWN trong $sortedVector$.

Hai vectors toàn cục $incomingSum[v]$ và $outgoingSum[v]$ được sử dụng để lưu số lượng tất cả các đỉnh đi và các đỉnh đến v đã được đưa vào danh sách đỉnh liền kề tương ứng $incomingEdges[v]$ và $outgoingEdges[v]$. Các tham số này sẽ được sử dụng để phục vụ cho việc lựa chọn chiều duyệt trong giải thuật bBFS để thực hiện các truy vấn tính khoảng cách ngắn nhất. Trong quá trình thi hành giải thuật 3.7, hai vectors này sẽ chưa được cập nhật lại mà chúng chỉ được cập nhật sau khi đã ghi nhận thực sự các cạnh thêm mới hay xoá đi trong G. Việc cập nhật sau này có thể khiến cho giá trị số đỉnh con và cháu khi xét lựa chọn hướng đi của bBFS không thực sự chính xác. Tuy nhiên, chúng ta sẽ chấp nhận các giá trị tương đối đó để đảm bảo các giá trị này là không đổi khi thi hành các truy vấn khoảng cách, từ đó cho phép song song hoá được tập lớn hơn các truy vấn khoảng cách.

Sau khi các truy vấn tính khoảng cách đã được thi hành xong, việc ghi nhận chính thức

trạng thái các cạnh được cập nhật trong *Updates* sẽ được thể hiện qua thuật toán 3.8 dưới đây. Các vectors *incomingSum* và *outgoingSum* lúc này mới được cập nhật lại các giá trị chính xác dựa vào các phép thêm/xoá cạnh trong *Updates*.

Thuật toán 3.8: akGroupPlus: Ghi nhận các phép toán cập nhật

```
1  Function CommitSerial(Updates)
       Input: Updates chứa các phép toán cập nhật (t, a, u, v); đồ thị G với một số cạnh
              có trạng thái UNKNOWN;
       Output: G và hai vectors incomingSum, outgoingSum được ghi nhận bởi
               Updates
2      foreach (t, a, u, v) ∈ Updates do
3          if a == 'A' then
4              CommitAdd(outgoingEdges[u], v) ;
5              outgoingSum[u]+ = outgoingEdge[v].size() ;
6              CommitAdd(incomingEdges[v], u) ;
7              incomingSum[v]+ = incomingEdges[u].size() ;
8          end
9          else
10             CommitDelete(outgoingEdges[u], v) ;
11             outgoingSum[u]− = outgoingEdge[v].size() ;
12             CommitDelete(incomingEdges[v], u) ;
13             incomingSum[v]− = incomingEdges[u].size() ;
14         end
15     end
```

Trong giải thuật này, các hàm *CommitAdd* và *CommitDelete* đảm nhiệm việc ghi nhận thực sự trạng thái ALIVE hay DEAD tương ứng với phép toán cập nhật. Việc thi hành hai hàm này cũng được tiến hành trên cả hai mảng vectors *outgoingEdges* và *incomingEdges*.

3.4.4 Tối ưu hoá các truy vấn tính khoảng cách ngắn nhất

3.4.4.1 Giải thuật tính khoảng cách ngắn nhất

Tương tự như giải thuật 3.4 tính khoảng cách ngắn nhất dựa trên phương pháp duyệt theo chiều rộng trước cả hai hướng (bBFS), việc xử lý các truy vấn tính khoảng cách ngắn nhất trong giải pháp này của chúng tôi cũng dựa trên bBFS. Chiến lược lựa chọn chiều duyệt cũng tương tự như trong giải thuật 3.4 [DPH1]. Ngoài ra, các mảng bitmaps *incomingMaps*/*outgoingMaps*) cũng được sử dụng trong mỗi luồng để lưu vết đỉnh đã duyệt trong bBFS. Việc xác định sử dụng hay không sử dụng các cạnh có trạng thái UNKNOWN hoàn toàn dựa vào thứ tự của truy vấn và được giao phó cho hàm $IsEdgeAlive(n, e, state, t)$

Với những tư tưởng đó, việc thi hành truy vấn tính khoảng cách ngắn nhất giữa hai đỉnh

u, v sẽ được minh hoạ bằng thuật toán 3.9 sau đây:

Thuật toán 3.9: akGroupPlus: Tính khoảng cách ngắn nhất giữa (u, v)

```
1  Function ComputeShortest2(t, u, v)
       Input: (t, u, v): truy vấn (u, v) tại thời điểm t; inMap, outMap: mảng để lưu vết duyệt; inQueue, outQueue:
              hàng đợi để lưu các đỉnh chuẩn bị duyệt; incomingSum, outgoingSum và đồ thị G
       Output: giá trị khoảng cách ngắn nhất từ u đến v
2      if u == v then return 0 ;
3      if (incomingEdges[v].size() == 0) || (outgoingEdges[u].size() == 0) then Return -1 /* không có đường đi từ
        u đến v */ ;
4      inQueue ← v; setBit(v, inMap) ;                                  /* khởi tạo inQueue và đánh dấu đã duyệt v */
5      outQueue ← u; setBit(u, outMap) ;                               /* khởi tạo outQueue và đánh dấu đã duyệt u */
6      inCost = 0, outCost = 0 ;                                              /* khởi tạo khoảng cách từ/đến u/v */
7      outSize = outgoingSum[u]; inSize = incomingSum[v] ;
8      while (outSize > 0)&&(inSize > 0) do
9          if (outSize < inSize) then                                  /* theo hướng duyệt các đỉnh đi outQueue */
10             outSize = 0; outCost+ = 1 ;
11             while outQueue not empty do
12                 e ← outQueue ;
13                 foreach n ∈ outgoingEdges[e] do
14                     if !testBit(n, outMap) then
15                         state = GetState(n) ;                                /* xác định trạng thái đỉnh u */
16                         if (state==ALIVE)||((state&UNKNOWN) && IsEdgeAlive(e,n,state,t)) then
17                             if testBit(n, inMap) then
18                                 Clear all bits of inMap & outMap; return inCost + outCost ;
19                             end
20                             outQueue ← n; setBit(n, outMap) ;
21                         end
22                     end
23                 end
24                 outSize+ = outgoingSum[e] ;
25             end
26         end
27         else                                                                /* theo hướng đến inQueue */
28             inSize = 0; inCost+ = 1 ;
29             while inQueue not empty do
30                 e ← inQueue ;
31                 foreach n ∈ incomingEdges[e] do
32                     if !testBit(n, inMap) then
33                         state = GetState(n) ;                                /* xác định trạng thái đỉnh u */
34                         if (state==ALIVE)||((state&UNKNOWN) && IsEdgeAlive(n,e,state,t)) then
35                             if testBit(n, outMap) then
36                                 Clear all bits of inMap & outMap; return inCost + outCost ;
37                             end
38                             inQueue ← n; setBit(n, inMap) ;
39                         end
40                     end
41                 end
42                 inSize+ = incomingSum[e] ;
43             end
44         end
45     end
46     Clear all bits of inMap & outMap ;
47     return-1 ;
```

Trong giải thuật này, hàm $setBit(v, map)$ đảm nhiệm gán giá trị 1 cho bit tại vị trí v trong mảng map. Trong khi đó hàm $testBit(v, map)$ trả về giá trị bit tại vị trí v trong mảng

map. Việc xác định trạng thái của một đỉnh u (dựa vào 2 bits cuối) sẽ được xác định thông qua hàm $GetState(u)$ (với cài đặt dựa trên các phép dịch bits để trả về 2 bits cuối).

Đối với các cạnh có giá trị trạng thái UNKNOWN (tức các cạnh đang được cập nhật thông qua các phép toán trong S), chúng ta sẽ dựa vào nhãn thời gian t của truy vấn so với các phép toán cập nhật trong $Updates$ để quyết định sử dụng hay không sử dụng cạnh đó. Ý tưởng đó sẽ được cài đặt trong hàm $IsEdgeAlive(u, v, t, state)$ như minh hoạ dưới đây:

Thuật toán 3.10: akGroupPlus: Kiểm tra một cạnh (u, v) có được ghi nhận ở thời điểm t hay không

```
1 Function IsEdgeAlive(u, v, t, state)
      Input: Bộ (u, v, state, t) gồm cạnh (u, v) với trạng thái hiện thời state tại thời
             điểm t; Updates
      Output: TRUE nếu (u, v) được ghi nhận tại thời điểm t; FALSE nếu không
2     i = lower_bound(Updates, (u, v, t, 0)) ;
3     if i == Updates.begin() then return (state&1) == 0 ;
4     (lu, lv, la, lt) = Updates[i − 1] ;
5     if (u == lu && v == lv) then return (la=='A') ;
6     return (state&1) == 0 ;
```

Trong giải thuật này, chúng ta sử dụng hàm *lower_bound* đã được cài đặt trong các thư viện chuẩn C++. Hàm này có chức năng tìm (dựa trên giải thuật tìm kiếm nhị phân) và trả về con trỏ *iterator* trỏ tới phần tử đầu tiên trong mảng đã được sắp xếp $Updates$ đảm bảo không nhỏ hơn bộ $(u, v, t, 0)$. Với trường hợp đầu tiên ($i == Updates.begin()$), cạnh (u, v) không xuất hiện trong tập các phép toán cập nhật $Updates$. Từ đó, cạnh này sẽ được sử dụng trong các phép toán tính khoảng cánh nếu ở trạng thái ALIVE hoặc UNKNOWN (tức $(state\&1) == 0$ trả về giá trị TRUE). Nếu tìm được bộ $(u, v, t, 0)$ trong $Updates$, cạnh (u, v) sẽ chỉ được sử dụng nếu như phép toán tương ứng với bộ đó trong $Updates$ là 'A' (phép thêm cạnh).

Như vậy, có thể khẳng định được giải thuật 3.9 cho phép tính được chính xác tại thời điểm thi hành trong lịch S khoảng cách ngắn nhất giữa u và v. Điều này cũng như kết quả thực nghiệm ở mục 3.6.3.3 cho phép khẳng định được tính đúng đắn của giải thuật này.

3.4.4.2 Xử lý song song truy vấn

Với tập các truy vấn đã được lưu trong vector $Queries$, cũng tương tự như đã phân tích trong mục trên, bộ thư viện CilkPlus được sử dụng để cài đặt tính toán song song các truy vấn trong $Queries$. Một điểm cũng đáng nhấn mạnh ở đây là chúng tôi vẫn sử dụng các mảng toàn cục $inMaps$ và $outMaps$ được cấp phát bộ nhớ trước với số bits tương ứng với số đỉnh nhân với số luồng có thể thi hành trên hệ thống tính toán. Khi đó, mỗi luồng thi hành các phép tính khoảng cách ngắn nhất sẽ hoạt động độc lập trên một vùng dữ liệu của các mảng toàn cục này. Ý tưởng này cũng được cài đặt tương tự với các hàng đợi $inQueue$

và *outQueue*.

Từ đó, việc xử lý song song các truy vấn khoảng cách trong *Queries* được tiến hành như trong giải thuật 3.11 dưới đây:

Thuật toán 3.11: akGroupPlus: Thi hành song song các truy vấn khoảng cách ngắn nhất trên đồ thị G

```
1 Function ParallelQuery(Queries)
      Input: Queries chứa các truy vấn khoảng cách (u, v) tại thời điểm t; đồ thị G
      Output: Danh sách Dist[.] chứa các khoảng cách tương ứng
      /* Thi hành song song vòng For sử dụng thư viện CilkPlus          */
2     for i = 0; i < Queries.size(); i++ do
3         (u, v, t) ← Queries[i] ;
4         Dist[i] = ComputeShortest2(u, v, t) ;
5     end
6     return Dist[.];
```

3.4.5 Đánh giá thuật toán

Việc thi hành lịch S theo giải pháp này vẫn đảm bảo được tính đúng đắn và tính nhất quán của đồ thị G. Thật vậy, mỗi phép cập nhật cạnh trong S luôn Trong giải pháp này, việc thi hành lịch S cũng luôn đảm bảo được tính đúng đắn cũng như tính nhất quán của đồ thị G. Thật vậy, mỗi phép toán cập nhật cạnh trong S luôn chuyển trạng thái cạnh đó về UNKNOWN. Tuy nhiên, trong quá trình tính khoảng cách ngắn nhất khi thi hành song song, việc cạnh đó có được sử dụng hay không (thông qua hàm $IsEdgeAlive$) sẽ được xác định dựa vào nhãn thời gian t (tương ứng với thứ tự phép toán cập nhật so với các phép toán tính khoảng cách ngắn nhất). Từ đó, các phép tính khoảng cách ngắn nhất luôn được thực hiện trên những dữ liệu đồ thị nhất quán và cho các kết quả như mong đợi.

Với việc tổ chức dữ liệu đồ thị cho phép nhúng luôn trạng thái mỗi cạnh vào trong 2 bits không được sử dụng (đa phần các đồ thị/mạng hiện nay đều có số đỉnh không vượt quá một tỷ, ngoại trừ Facebook), việc tiến hành các phép toán cập nhật được triển khai tương đối thuận lợi so với giải pháp 1. Trong các phép toán đó, công việc chính là tìm đến vị trí danh sách đỉnh liền kề đã được sắp xếp tăng dần (độ phức tạp $O(log(n))$) và cập nhật trạng thái. Việc không xoá thực sự đỉnh trong danh sách liền kề giúp tránh phải tổ chức lại danh sách đó. Ngoài ra, việc tổ chức theo cách thức này cũng mang lại hiệu quả đối với các phép toán xoá cạnh rồi sau đó lại thêm lại chính cạnh đó chẳng hạn.

Đối với giải thuật tìm đường đi ngắn nhất, độ phức tạp của $ComputeShortest2$ trong trường hợp tồi nhất vẫn là $O(|V| + |E|)$. Tuy nhiên, cũng giống như trong giải pháp 1, nếu tính trung bình mỗi đỉnh có b cạnh đến/đi và khoảng cách ngắn nhất trung bình là k, khi đó độ phức tạp trung bình của giải thuật này vẫn là $O(b^{\frac{k}{2}})$.

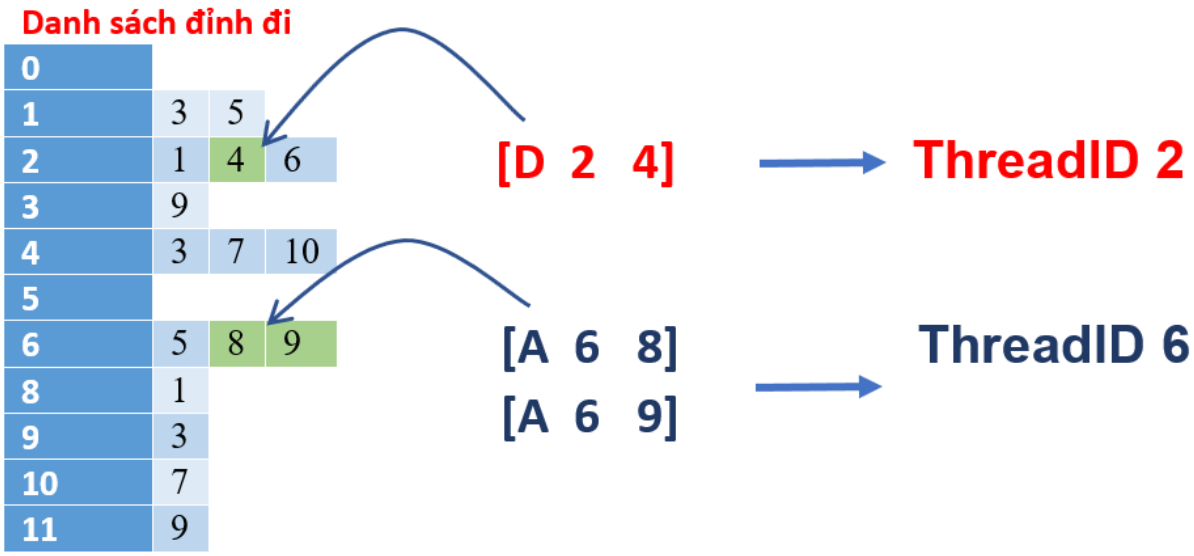
Với việc sử dụng trạng thái UNKNOWN để đánh dấu các cạnh cần phải cập nhật trong

lịch S, quá trình xử lý các truy vấn, lúc bấy giờ, có thể được tiến hành song song với toàn bộ các truy vấn trong S. Đây cũng là ưu điểm của phương pháp này so với phương pháp 1 nêu trên (có thể bị ngắt quãng khi gặp các phép toán cập nhật). Và khi tiến hành thực nghiệm, hiệu năng của giải pháp này tốt hơn so với akGroup trong giải pháp 1 (chi tiết được trình bày trong mục 3.6 của chương này).

3.5 Giải pháp 3: bigGraph

3.5.1 Ý tưởng chính

Dựa trên giải pháp akGroupPlus, chúng tôi nhận thấy là ngay cả các phép toán cập nhật cũng có thể được thi hành song song nếu đảm bảo các thao tác cập nhật trên danh sách liền kề $outgoingEdges[v_t]/incomingEdges[v_t]$ phải được thi hành tuần tự trên một luồng t riêng biệt. Từ đó, nếu chúng ta thi hành song song $threadNum$ luồng, chúng ta có thể tiến hành xử lý đồng thời $threadNum$ phép toán cập nhật trên $threadNum$ danh sách các đỉnh liền kề có định danh khác nhau mà không cần phải sử dụng bất kỳ kỹ thuật kiểm soát tương tranh nào. Ý tưởng này được minh hoạ như trong hình 3.7 dưới đây:



Hình 3.7: Song song các phép toán cập nhật đồ thị

Trong hình 3.7, đối với các phép toán cập nhật danh sách đỉnh đi 6, chúng ta cần phải thi hành tuần tự để đảm bảo tính nhất quán của đồ thị G (hoặc không phải xử lý tương tranh). Tuy nhiên, rõ ràng các phép toán cập nhật danh sách các đỉnh đi của đỉnh 2 và 6 có thể tiến hành song song độc lập với nhau mà không ảnh hưởng đến tính nhất quán của G.

Với ý tưởng đó, giải pháp bigGraph được xây dựng hoàn toàn dựa trên giải pháp 2, nhưng bổ sung thêm kỹ thuật song song các phép toán cập nhật. Khi đó, lịch thi hành các phép toán tương tranh S sẽ được xử lý theo các bước sau:

- Lưu lại có thứ tự các phép toán cập nhật vào mảng Updates; các phép toán truy vấn đường đi ngắn nhất vào mảng Queries;
- Tiến hành xử lý *song song* các phép toán cập nhật và đánh dấu các cạnh được cập nhật về trạng thái UNKNOWN với ý tưởng: mở $numThreads$ luồng đồng thời; việc cập nhật danh sách đỉnh liền kề của v chỉ được phép tiến hành trên luồng thứ v mod $numThreads$.
- Tiến hành xử lý song song các truy vấn đường đi ngắn nhất;
- Cập nhật chính thức trạng thái các cạnh liên quan đến các phép toán cập nhật, tức chuyển trạng thái các cạnh đó từ UNKNOWN về DEAD (nếu bị xoá) hay ALIVE (nếu được thêm vào) theo phương pháp xử lý *song song* với cùng ý tưởng nêu trên;

Các bước xử lý trên có thể được minh hoạ cụ thể hơn qua giải thuật 3.12 dưới đây:

Thuật toán 3.12: bigGraph: Giải thuật thi hành lịch S

```
1  Function bigGraph(S)
       Input: Đồ thị G và danh sách S có n phép toán (a, u, v) với 'a' là phép toán
       Output: G ghi nhật tất cả cập nhật và danh sách khoảng cách ngắn nhất Dist[.]
               của tất cả truy vấn 'Q'
2      for t=0; t < n; t++ do
3          (a,u,v) = S[t] ;
4          if a =' Q' then
5              Queries.push_back(t,u,v) ;      /* đẩy bộ (t, u, v) vào cuối Queries */
6          end
7          else
8              Updates.push_back(t,a,u,v) ;    /* đẩy (t, a, u, v) vào cuối Updates */
9          end
10     end
11     UpdateParallel(Updates) ;   /* Cập nhật trạng thái các cạnh theo giải
           thuật 3.13  */
12     Dist[] = ParallelQuery(Queries) ;   /* Thi hành song song các truy vấn
           tính khoản cách ngắn nhất 'Q' theo giải thuật 3.11  */
13     CommitParallel(Updates) ;      /* Ghi nhận các cạnh thêm/xoá trong G
           theo giải thuật 3.14  */
14     return Dist[.] ;
```

Như vậy, trong giải pháp này, chúng ta sẽ dùng lại toàn bộ cách tiếp cận xử lý song song các truy vấn trong giải pháp akGroupPlus.

3.5.2 Giải thuật song song hoá các phép toán cập nhật

Như đã phân tích ở trên, các phép toán cập nhật sẽ được song song hoá dựa trên ý tưởng: mở $numThreads$ luồng đồng thời; việc cập nhật danh sách đỉnh liền kề của v chỉ được phép tiến hành trên luồng thứ v mod $numThreads$.

Các giải thuật cập nhật mới sẽ được trình bày cụ thể dưới đây:

Thuật toán 3.13: bigGraph: Song song hoá các phép toán cập nhật trong S

```
1  Function UpdateParallel(Updates)
       Input: Updates: vector chứa các phép toán cập nhật kèm nhãn thời gian (t, a, u, v); đồ thị G; threadNum là số
              luồng tối đa muốn thi hành song song các phép cập nhật
       Output: G được cập nhật trạng thái một số cạnh bởi Updates
       /* Thi hành song song vòng For sử dụng thư viện CilkPlus với outgoingEdges                    */
2      for w = 0; w < threadNum; w++ do
3          foreach (t, a, u, v) ∈ Updates do
4              if u mod threadNum == w then
5                  if a == 'A' then
6                      InsertNode(outgoingEdges[u], v) ;
7                  end
8                  else
9                      RemoveNode(outgoingEdges[u], v) ;
10                 end
11             end
12         end
13     end
       /* Thi hành song song vòng For sử dụng thư viện CilkPlus với incomingEdges                    */
14     for w = 0; w < threadNum; w++ do
15         foreach (t, a, u, v) ∈ Updates do
16             if v mod threadNum == w then
17                 if a == 'A' then
18                     InsertNode(incomingEdges[v], u) ;
19                 end
20                 else
21                     RemoveNode(incomingEdges[v], u) ;
22                 end
23             end
24         end
25     end
```

Trong giải thuật này, hai vòng For lồng nhau đảm nhiệm việc mở $threadNum$ luồng khác nhau, và mỗi phép cập nhật đỉnh v chỉ được tiến hành trong luồng có định danh w nếu đảm bảo $vmodthreadNum == w$. Với ràng buộc đó, hai vòng lặp này rõ ràng chỉ cho phép thực hiện duy nhất một lần cập nhật mỗi bộ trong $Updates$. Điều này cho phép khẳng định được tính đúng đắn của giải thuật này khi thi hành các phép toán cập nhật trên đồ thị G.

Còn đối với việc ghi nhận thực sự các phép toán cập nhật sau khi đã tính xong các truy vấn tính khoảng cách ngắn nhất, hàm $CommitParallel$ được minh hoạ như giải thuật 3.14

dưới đây:

Thuật toán 3.14: bigGraph: Ghi nhận các phép toán cập nhật

```
1  Function CommitParallel(Updates)
       Input: Updates chứa các phép toán cập nhật (t, a, u, v); đồ thị G với một số cạnh
              có trạng thái UNKNOWN;
       Output: G và hai vectors incomingSum, outgoingSum được ghi nhận bởi
               Updates
       /* Thi hành song song vòng For sử dụng thư viện CilkPlus          */
2      foreach (t, a, u, v) ∈ Updates do
3          if a == 'A' then
4              CommitAdd(outgoingEdges[u], v) ;
5              CommitAdd(incomingEdges[v], u) ;
6          end
7          else
8              CommitDelete(outgoingEdges[u], v) ;
9              CommitDelete(incomingEdges[v], u) ;
10         end
11     end
12     foreach (t, a, u, v) ∈ Updates do
13         if a == 'A' then
14             outgoingSum[u]+ = outgoingEdge[v].size() ;
15             incomingSum[v]+ = incomingEdges[u].size() ;
16         end
17         else
18             outgoingSum[u]− = outgoingEdge[v].size() ;
19             incomingSum[v]− = incomingEdges[u].size() ;
20         end
21     end
```

Trong giải thuật này, chúng ta tiến hành song song các phép toán ghi nhận cập nhật trên G thông qua các hàm *CommitAdd* và *CommitDelete* giống như trong giải pháp 2 akGroupPlus. Tuy nhiên, việc cập nhật các giá trị tổng số đỉnh con được tách riêng để tránh phải xử lý tương tranh. Các phép toán này thực hiện tương đối nhanh nên được tiến hành tuần tự mà không cần phải xử lý song song.

3.5.3 Đánh giá thuật toán

Đối với giải pháp này, rõ ràng việc thi hành lịch S cũng cho kết quả đúng đắn và đảm bảo được tính nhất quán của đồ thị G. Thật vậy, giải pháp bigGraph dựa hoàn toàn trên

akGroupPlus, chỉ khác biệt ở khả năng song song hoá cả quá trình thi hành các phép toán cập nhật. Dựa trên cấu trúc dữ liệu đồ thị dạng liên kết liền kề, việc thi hành cập nhật cạnh (u, v) rõ ràng chỉ ảnh hưởng đến các mảng $outgoingEdges[u]$ và $incomingEdges[v]$. Từ đó, nếu các phép toán cập nhật đồng thời cần phải tiến hành trên nhiều đỉnh khác nhau thì chúng có thể thi hành song song mà không ảnh hưởng đến tính nhất quán của dữ liệu đồ thị G.

Về mặt lý thuyết, giải pháp bigGraph có độ phức tạp tính toán tương đương với giải pháp akGroupPlus. Tuy nhiên, với việc bổ sung thêm phương pháp song song hoá các phép toán cập nhật, thời gian thi hành các phép toán tương tranh trong S chắc chắn sẽ hiệu quả hơn so với akGroupPlus. Thực nghiệm trong mục 3.6 của chương này đã minh chứng khẳng định này.

3.6 Thực nghiệm và đánh giá

Trong phần này, chúng tôi sẽ trình bày các kết quả thực nghiệm đánh giá các giải pháp mô hình hoá các phép toán tương tranh trên đồ thị đã được trình bày ở trên.

3.6.1 Môi trường và dữ liệu thực nghiệm

3.6.1.1 Môi trường thử nghiệm, đánh giá

Ba giải pháp đã trình bày trong chương 2 đã được chúng tôi tiến hành thử nghiệm đánh giá trên nhiều môi trường tính toán khác nhau, cụ thể như sau:

- Giải pháp akGroup đã tham gia cuộc thi ACM SigMod Programming Contest 2016, được thực nghiệm trên hệ thống tính toán với cấu hình 2 x Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz (45MB Cache, 18-cores per CPU), bộ nhớ chính 128GB, CentOS Linux release 7.2.1511, gcc 6.3.0. Giải pháp akGroup đã được cài đặt sử dụng ngôn ngữ C chuẩn.

- Chúng tôi cũng tiến hành thử nghiệm đánh giá giải pháp akGroup với các bộ dữ liệu khác, trên máy tính cá nhân với cấu hình Intel® Core™ i7-3720QM (6MB Cache, up to 3.60 GHz, 4 cores-8 threads), bộ nhớ 8GB, CentOS 7, gcc 5.1.1.

- Với các thử nghiệm, đánh giá các giải pháp 2 và 3, chúng tôi đã tiến hành trên hệ thống tính toán hiệu năng cao của Trường Đại học Công nghệ, với cấu hình 2 x Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz (45MB Cache, 18-cores per CPU), bộ nhớ chính 128GB, CentOS Linux release 7.2.1511, gcc 6.3.0. Hệ thống tính toán này được cấu hình cho phép thi hành tối đa 36 luồng song song (do cấu hình tắt chức năng hyperthreading trên 2 CPU). Các giải pháp akGroupPlus và bigGraph đều được cài đặt sử dụng ngôn ngữ C++.

3.6.1.2 Dữ liệu thực nghiệm

Các giải pháp xử lý phép toán tương tranh trên đồ thị đều được chúng tôi tiến hành với những bộ dữ liệu đã được công bố từ các tổ chức có uy tín khoa học trên thế giới.

3.6.1.2.1 Dữ liệu từ cuộc thi SigMod Programming Contest 2016

Dữ liệu của cuộc thi này là 4 đồ thị có hướng không trọng số quy mô lớn được sử dụng để đánh giá kết quả của 33 đội tham gia. Thông tin chi tiết về 4 đồ thị này được tổng hợp trong bảng dưới đây:

Bảng 3.2: Thống kê các đồ thị sử dụng trong SigMod 2016

	Tập thử nhỏ	**Tập thử trung bình**	**Tập thử lớn**	**Tập thử rất lớn**
Số đỉnh	1.574.074	2.000.000	1.971.281	6.009.555
Số cạnh	3.232.855	5.000.000	5.533.214	16.518.948

Trong các tập thử này, ngoài những tham số liên quan đến số đỉnh, số cạnh đồ thị, quy mô tập thử còn phụ thuộc vào số lượng phép toán đồng thời dùng trong mỗi tập thử. Các tham số liên quan đến số phép toán thêm, xoá cạnh cũng như số truy vấn khoảng cách ngắn nhất không được công bố công khai trong cuộc thi này.

3.6.1.2.2 Dữ liệu SNAP

Ngoài các bộ dữ liệu của SigMod, chúng tôi cũng đã tiến hành lựa chọn và thu thập các bộ dữ liệu đồ thị nổi tiếng khác để tiến hành thử nghiệm. Hiện nay, các bộ dữ liệu đồ thị được trường Đại học Stanford công bố vẫn được xem là những bộ dữ liệu điển hình để đánh giá các công cụ xử lý đồ thị [62]. Các bộ dữ liệu này được tập hợp gọi tắt là SNAP (Stanford Large Network Dataset Collection) [61]. Qua khảo sát, chúng tôi đã chọn hai bộ dữ liệu chính là **Pokec** và **LiveJournal** để tiến hành các thực nghiệm đánh giá. Bảng sau minh hoạ các đặc điểm chính của hai bộ dữ liệu này cùng với bộ dữ liệu SigMod đã được công bố công khai:

Bảng 3.3: Thống kê các bộ dữ liệu đồ thị sử dụng trong thực nghiệm

Tham số	**SigMod**	**Pokec**[3]	**LiveJournal**[4]
Số cạnh	1.574.074	68.993.773	30.622.564
Số đỉnh	3.232.855	4.847.571	1.632.803
Đường kính (khoảng cách ngắn nhất lớn nhất)	9	16	11

Trong SNAP, bộ dữ liệu LiveJournal được tập hợp từ một mạng xã hội miễn phí trực tuyến hiện có hơn 39 triệu thành viên của Nga (nhưng đặt trụ sở chính ở San Francisco,

Mỹ) [63]. Mạng xã hội LiveJournal cho phép các thành viên duy trì các bài viết, blog cá nhân và nhóm; cho phép các thành viên xác lập quan hệ bạn bè với thành viên khác. Tập dữ liệu LiveJournal của SNAP đã tập hợp được hơn 4,8 triệu thành viên với hơn 68 triệu quan hệ (đã được xử lý ẩn danh). Trong khi đó, Pokec cũng là một mạng xã hội trực tuyến nổi tiếng tại Slovakia, ngay cả khi đã có Facebook [100]. Trong bộ dữ liệu Pokec đã được xử lý ẩn danh người dùng công bố trong SNAP, có 1,6 triệu người tham gia với hơn 30 triệu kết nối quan hệ.

3.6.2 Phương pháp thử nghiệm, đánh giá

Trong các thực nghiệm chúng tôi tiến hành, đồ thị có hướng không trọng số G ban đầu sẽ được cung cấp dưới dạng một tệp văn bản, cấu trúc như sau:

- Mỗi đỉnh sẽ được định danh bằng một số tự nhiên có giá trị tối đa là $2^{30} - 1$;
- Mỗi cạnh trong đồ thị được lưu trên một hàng trong tệp, phân cách hoặc bằng ký tự cách, hoặc ký tự tab.
- Các thông tin mô tả có đưa vào tệp dữ liệu, tuy nhiên phải bắt đầu bằng ký tự "#".

3.6.2.1 Sinh các tập lịch thi hành thử nghiệm

Để thuận lợi cho việc kiểm tra và đánh giá các kết quả thực nghiệm, tập các lịch thi hành tương tranh S cũng sẽ được lưu trong một tệp văn bản. Mỗi dòng sẽ thể hiện một phép toán op_i thể hiện bộ (auv) với a là ký tự tương ứng với phép toán thêm 'A', xoá 'D', hay truy vấn khoảng cách ngắn nhất 'Q'. Để đánh dấu kết thúc một lịch S, chúng tôi sử dụng ký tự F (Finish) trên một hàng riêng. Phương pháp tổ chức lịch thi hành này cũng tương tự như khái niệm *lô (batch)* đưa ra trong cuộc thi ACM SigMod Programming Contest 2016 [97].

Để có các tập dữ liệu chứa các lịch thi hành S thử nghiệm (gọi tắt là workload), chúng tôi đã xây dựng công cụ theo đúng quy định đã nêu trên. Tập các lịch thi hành được sinh với 1.000.000 phép toán bao hàm cả các phép toán cập nhật lẫn truy vấn khoảng cách ngắn nhất. Thực tế, phân bố số lượng các phép toán thêm/xoá cạnh/truy vấn thường khác nhau, nhưng đa phần là chúng ta cần phải xử lý các phép truy vấn nhiều hơn so với cập nhật. Chính vì thế, chúng tôi đã sinh ra hai tập workloads như sau:

- Tập workload có số truy vấn khoảng cách nhiều hơn sẽ có phân bố các phép toán Q/A/D lần lượt là 0,8/0,1/0,1; tức trong 1.000.000 phép toán của tệp dữ liệu này sẽ có 800.000 truy vấn khoảng cách ngắn nhất 'Q'; 100.000 phép toán thêm cạnh 'A' và 100.000 phép toán xoá cạnh 'D'. Tập workload này gọi tắt là tập 8-1-1.
- Tập workload có số lượng phép toán cập nhật lớn hơn sẽ có phân bố các phép toán Q/A/D lần lượt là 0,5/0,4/0,1; tức trong 1.000.000 phép toán của tệp dữ liệu này sẽ

có 500.000 truy vấn khoảng cách ngắn nhất 'Q'; 400.000 phép toán thêm cạnh 'A' và 100.000 phép toán xoá cạnh 'D'. Tập workload này gọi tắt là tập 5-4-1.

Lý do lựa chọn tỷ lệ 0,5/0,4/0,1 của chúng tôi xuất phát từ nhận xét đa phần với các đồ thị thực tế, đặc biệt là đối với các bài toán mô hình hoá mạng xã hội thực tế, các phép toán xoá cạnh (tức loại bỏ quan hệ) tương đối ít so với các phép thêm cạnh [18][109].

Từ các tập dữ liệu workload này, thông qua hai bộ công cụ NetworkX [81] và SNAP C++ [60], chúng tôi đã cho tiến hành chạy tuần tự tập các phép toán tương tranh trong các tệp workloads nêu trên và sinh ra các tệp kết quả tương ứng (khi thi hành, các bộ công cụ NetworkX và SNAPC++ đều cho cùng một kết quả đối với mỗi tập workload). Các tệp kết quả này sẽ được lưu lại làm cơ sở để kiểm tra tính đúng đắn của các phương pháp được xây dựng trong các mục trước.

3.6.2.2 Phương pháp đo

Việc đo hiệu năng cả ba giải pháp mà chúng tôi đã trình bày cũng như các giải pháp khác phục vụ so sánh, đánh giá đều được tiến hành trên cùng một môi trường thực nghiệm và cùng bộ dữ liệu thực nghiệm.

Để đánh giá hiệu năng thi hành của ba giải pháp nên trên, về cơ bản chúng tôi sẽ tiến hành cài đặt trên các hệ thống tính toán, từ đó thay đổi tham số về số luồng đồng thời để xác định các hệ số tăng tốc (speedup) [102][DPH3]. Với các giải pháp 2 và 3 khi chúng tôi đã có hệ thống tính toán hiệu năng cao, số luồng đồng thời được tiến hành thực nghiệm sẽ tăng lần lượt từ 1, 2, 4, 8, 16, 24, và 32 luồng (do hệ thống tính toán có tối đa 36 luồng). Để tránh những tác động từ các tiến trình khác ảnh hưởng đến kết quả thử nghiệm, mỗi khi thực nghiệm một giải pháp nào, chúng tôi tiến hành lặp lại 10 lần và sau đó lấy giá trị thời gian trung bình để đánh giá giải pháp đó.

Việc đo hiệu năng của các giải pháp trong các thực nghiệm của chúng tôi được tiến hành với công cụ *harness* được nhóm nghiên cứu "Data System Group" của trường Đại học Waterloo xây dựng năm 2015 để đo thời gian thi hành các giải pháp xử lý truy vấn trong các cuộc thi của ACM SigMod Contest [5]. Công cụ này có cú pháp *harness* <*init-graph*> <*workload*> <*result*> <*solution*>, trong đó <*init-graph*> là tệp chứa dữ liệu đồ thị ban đầu; <*workload*> là tệp chứa các lịch thi hành truy vấn tương tranh; <*result*> là tệp chứa kết quả các truy vấn tương ứng với các lịch thi hành trong <*workload*>; và <*solution*> là tệp thi hành tương ứng với giải pháp cần đo. Công cụ *harness* này sử dụng cấu trúc *timeeval* và hàm *gettimeofday* trong thư viện chuẩn *sys/time.h* để đo và trả về thời gian thi hành của mỗi giải pháp trên toàn bộ hệ thống tính toán với đơn vị mili giây.

Một điểm cũng cần nhấn mạnh thêm là công cụ *harness* cũng đảm nhiệm chức năng đánh giá tính đúng đắn của giải thuật cần đánh giá dựa vào tập kết quả chính thức đã sinh từ trước lưu trong tệp <*result*>. Trong quá trình đánh giá, giải thuật sinh ra bất kỳ kết quả

[5]Tham khảo tại trang `https://uwaterloo.ca/data-systems-group/`

sai khác so với kết quả chính thức, công cụ đánh giá này sẽ hiển thị thông báo lỗi tương ứng với truy vấn đó và dừng thi hành quá trình đánh giá.

3.6.3 Thử nghiệm và đánh giá kết quả

Dựa trên môi trường thực nghiệm, dữ liệu đã thu thập với các workloads sinh ngẫu nhiên và phương pháp thực nghiệm nêu trên, chúng tôi đã tiến hành cài đặt và thực nghiệm cả ba giải pháp xử lý các truy vấn khoảng cách ngắn nhất đã trình bày ở trên. Toàn bộ các kết quả thực nghiệm thu được dựa trên công cụ đánh giá *harness* đều minh chứng cả ba giải pháp đã đề xuất (akGroup, akGroupPlus và bigGraph) đều cho kết quả thực hiện các truy vấn đúng, đảm bảo được tính nhất quán khi thi hành các lịch thi hành đồng thời. Điều này cũng cho phép khẳng định thêm tính đúng đắn của các giải thuật đã đề xuất trong luận án (đã được chứng minh lý thuyết ở các mục trên). Do đó, trong các mục tiếp theo, chúng tôi sẽ không đề cập đến kết quả đảm bảo tính đúng đắn của các giải thuật này nữa.

3.6.3.1 Kết quả từ cuộc thi ACM SigMod Programming Contest 2016

Với giải pháp 1, nhóm akGroup của chúng tôi đã tham gia cuộc thi ACM SigMod Programming Contest năm 2016 và đã được giải Ba tại cuộc thi đó. Kết quả các lần chạy tốt nhất trên nền tảng hệ thống thử nghiệm của cuộc thi với cấu hình 2 x Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz (45MB Cache, 18-cores per CPU), bộ nhớ chính 128GB, CentOS Linux release 7.2.1511, gcc 6.3.0 được trình bày trong bảng dưới đây (bao gồm cả kết quả của đội nhất và nhì):

Bảng 3.4: Kết quả thực nghiệm trên hệ thống đánh giá của ACM SigMod 2016 (giây)

Đội tham dự	**Tập thử nhỏ**	**Tập thử trung bình**	**Tập thử lớn**	**Tập thử rất lớn**
H_minor_free	**0,107**	0,220	**0,886**	2,333
uateam	0,202	**0,217**	1,085	**2,123**
akGroup	0,118	0,494	1,284	2,878

3.6.3.2 Đánh giá giải pháp akGroup

Ngoài việc tham gia cuộc thi trên, để đánh giá giải pháp akGroup, chúng tôi đã tiến hành thực nghiệm, so sánh với một số phương pháp như chỉ dùng BFS thông thường đối với truy vấn khoảng cách (gọi tắt là BFS+CilkPlus (vẫn dùng thư viện CilkPlus để tiến hành song song)); bộ công cụ xử lý các phép toán tương tranh sử dụng NetworkX và Python[6]; bộ công cụ sử dụng thư viện SNAP C++ [7]. Workload được chúng tôi sử dụng trong thực nghiệm này là bộ dữ liệu có đa phần là truy vấn khoảng cách, tức bộ 8-1-1 với 1.000.000

[6] `http://dsg.uwaterloo.ca/sigmod16contest/reference.tar.gz`

[7] `https://snap.stanford.edu/releases/Snap-2.4.zip`

phép toán cho cả ba bộ dữ liệu đã nêu trên. Kết quả thực nghiệm được tổng hợp với 10 lần chạy khác nhau trên hệ thống tính toán là máy tính xách tay cấu hình Intel® Core™ i7-3720QM (6MB Cache, up to 3.60 GHz, 4 cores-8 threads), bộ nhớ 8GB, CentOS 7, gcc 5.1.1. Bảng 3.5 dưới đây minh hoạ các kết quả thực nghiệm mà chúng tôi đã thu được:

Bảng 3.5: Kết quả đánh giá giải pháp akGroup so với một số công cụ khác

Tập dữ liệu	**akGroup**	**BFS+CilkPlus**	**NetworkX**	**SNAP C++**
SigMod Test	**1,428s**	18,078s	3319,4s	5045s
LiveJournal	**32,045s**	55,630s	Không thể thi hành	>24h
Pokec	**14,286s**	27,825s	>10h	>15h

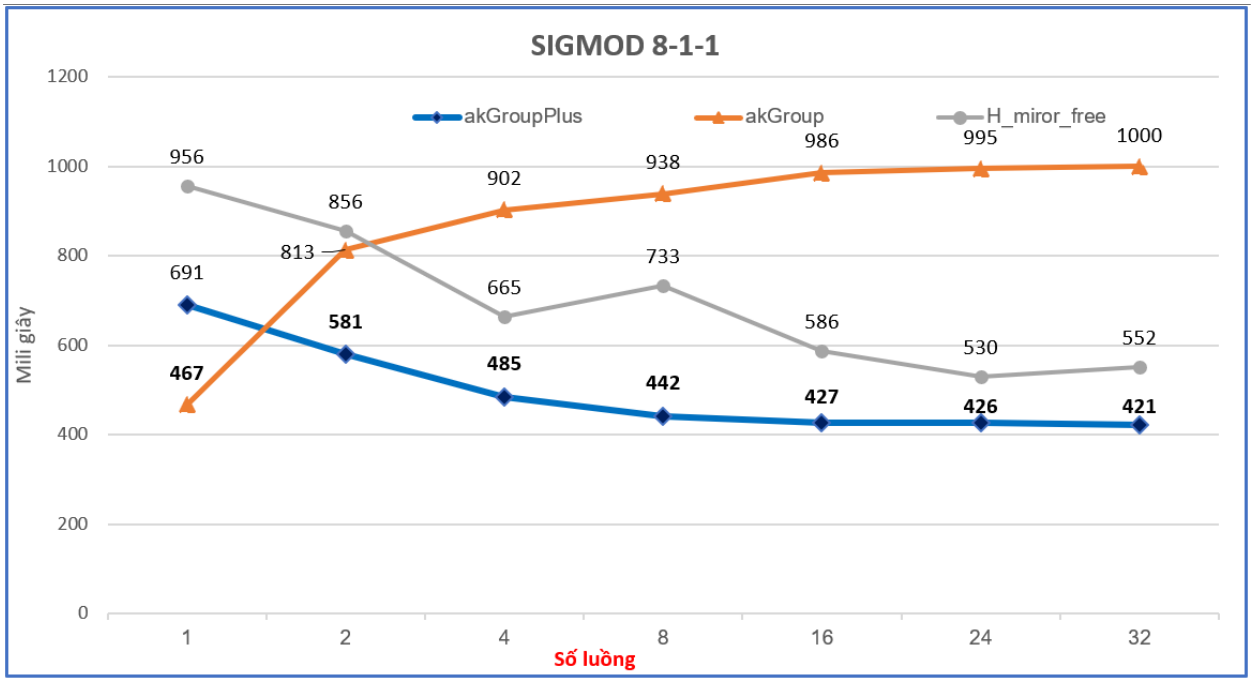
Dựa trên bảng kết quả này có thể thấy giải pháp akGroup nhìn chung đã mang lại hiệu quả thi hành tốt hơn so với các giải pháp khác được thử nghiệm. Điều này minh chứng việc sử dụng cấu trúc dữ liệu đồ thị phù hợp, chiến lược lựa chọn hướng duyệt BFS theo cả hai chiều và phương pháp song song hoá sử dụng CilkPlus của chúng tôi đã cho phép rút ngắn thời gian thi hành các phép toán tương tranh trên đồ thị. Các kết quả thi hành cũng như toàn bộ mã nguồn của akGroup đều có thể truy cập công khai tại địa chỉ `https://github.com/hanhdp/shortest_path`. Đây cũng là kết quả mà chúng tôi đã công bố trong công trình [DPH1] tại hội thảo BDCAT về quản lý dữ liệu lớn năm 2016.

3.6.3.3 Đánh giá giải pháp akGroupPlus

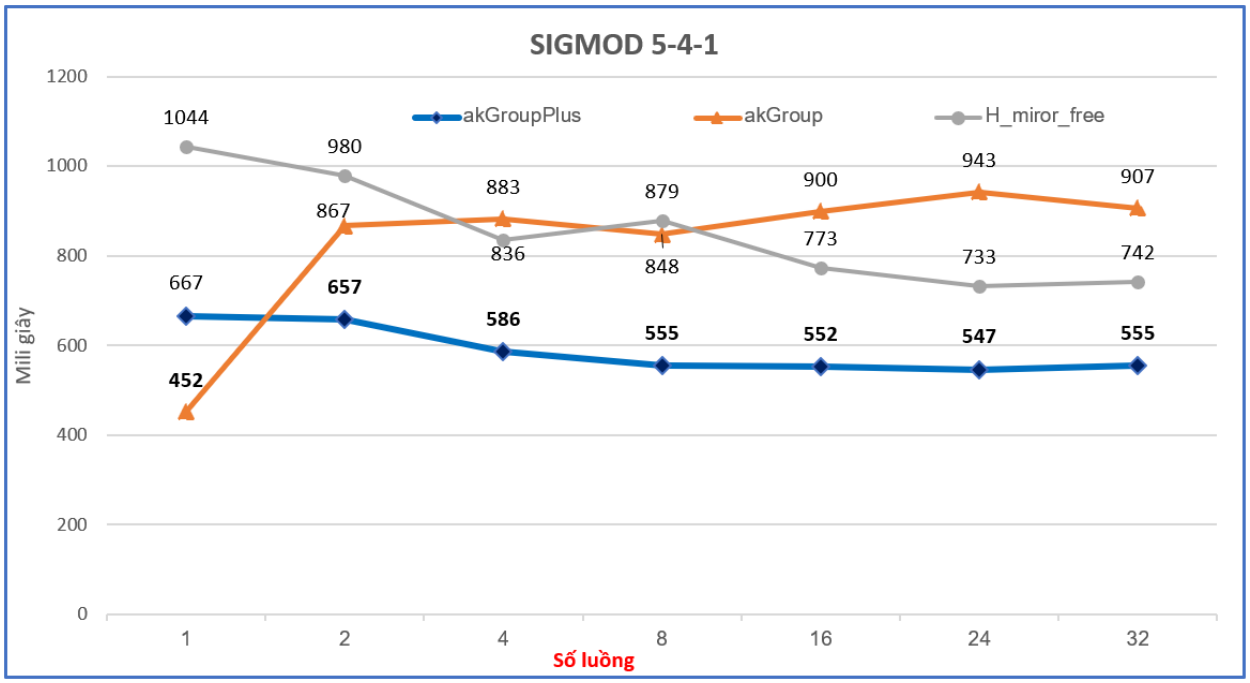
Bởi vì giải pháp akGroup vẫn còn chưa hiệu quả đối với các phép toán cập nhật, và cũng dựa trên các kết quả từ các nhóm khác tham gia cuộc thi ACM SigMod Contest 2016, chúng tôi đã tiến hành nghiên cứu mở rộng giải pháp này để hình thành giải pháp mới mang tên akGroupPlus. Giải pháp thứ hai này được xây dựng dựa trên phương pháp tổ chức dữ liệu kiểu vector đỉnh liền kề, trong đó có nhúng thông tin trạng thái cạnh vào 2 bits cuối của mỗi đỉnh. Từ đó cho phép nâng cao được hiệu năng của cả phép toán cập nhật lẫn truy vấn khoảng cách.

Để đánh giá hiệu năng **akGroupPlus**, chúng tôi đã tiến hành thực nghiệm so sánh giải pháp này với hai giải pháp khác là: **akGroup** - giải pháp cũ đã trình bày ở trên; và **H_minor_free** giải pháp đã được giải nhất trong cuộc thi ACM SigMod Contest 2016.

Quá trình thực nghiệm cũng vẫn được tiến hành với cả hai bộ dữ liệu 8-1-1 và 5-4-1 và với các đồ thị SigMod, Pokec và Livejournal. Số lượng luồng được thay đổi từ 1, 2, 4, 8, 16, 24, và 32 luồng để đánh giá độ tăng tốc của mỗi giải thuật (hệ thống tính toán hiệu năng cao chỉ có tối đa 36-luồng). Mỗi một giải pháp sẽ được thực thi lặp lại 10 lần và lấy kết quả trung bình. Kết quả thực nghiệm được minh họa qua các hình dưới đây:

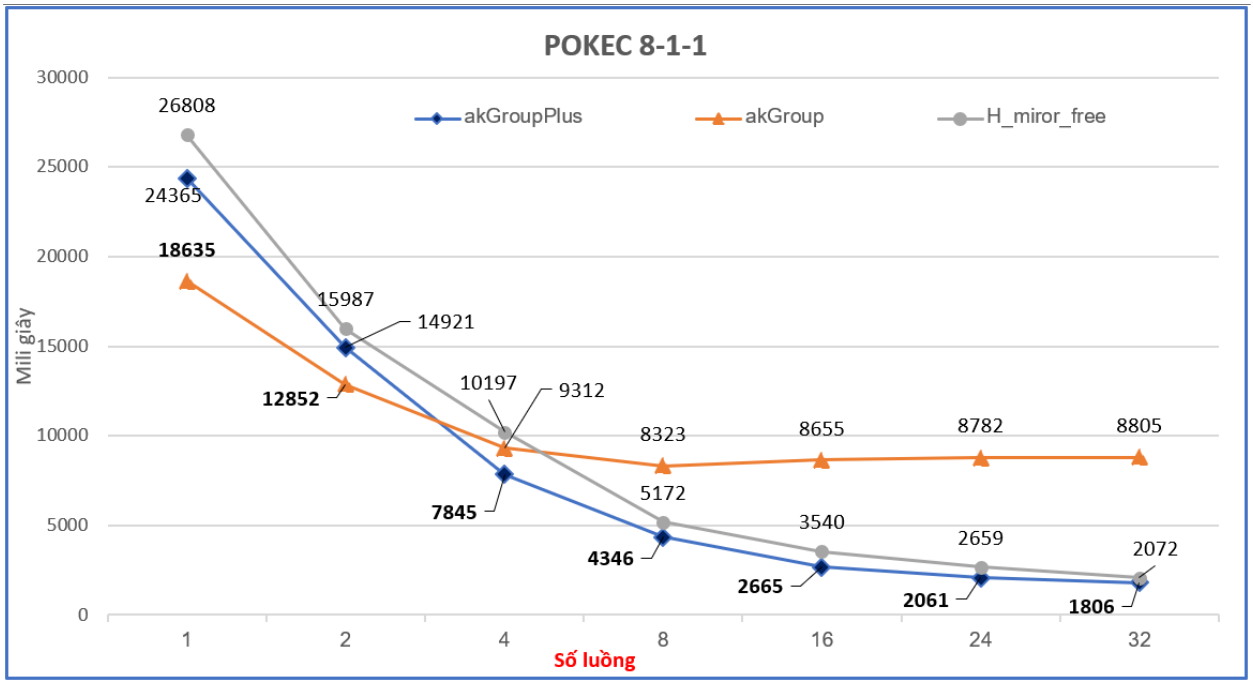


(a) Kết quả đánh giá với tập 8-1-1

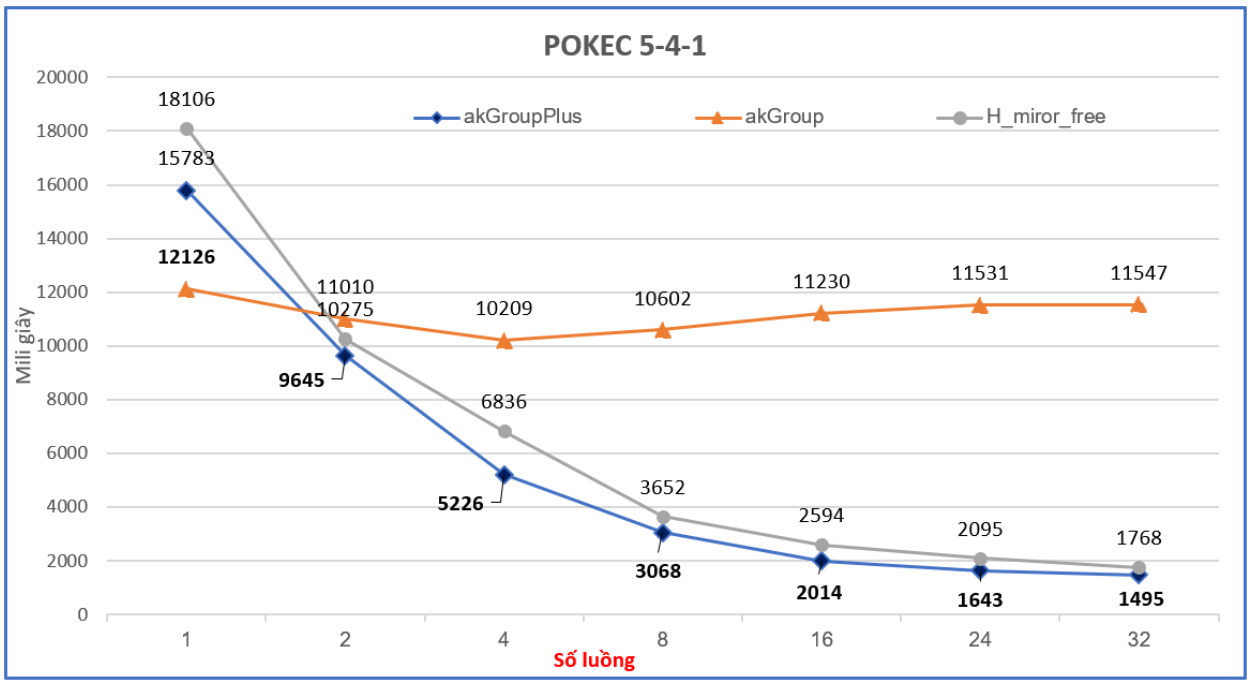


(b) Kết quả đánh giá với tập 5-4-1

Hình 3.8: Kết quả đánh giá với bộ dữ liệu Sigmod Dataset

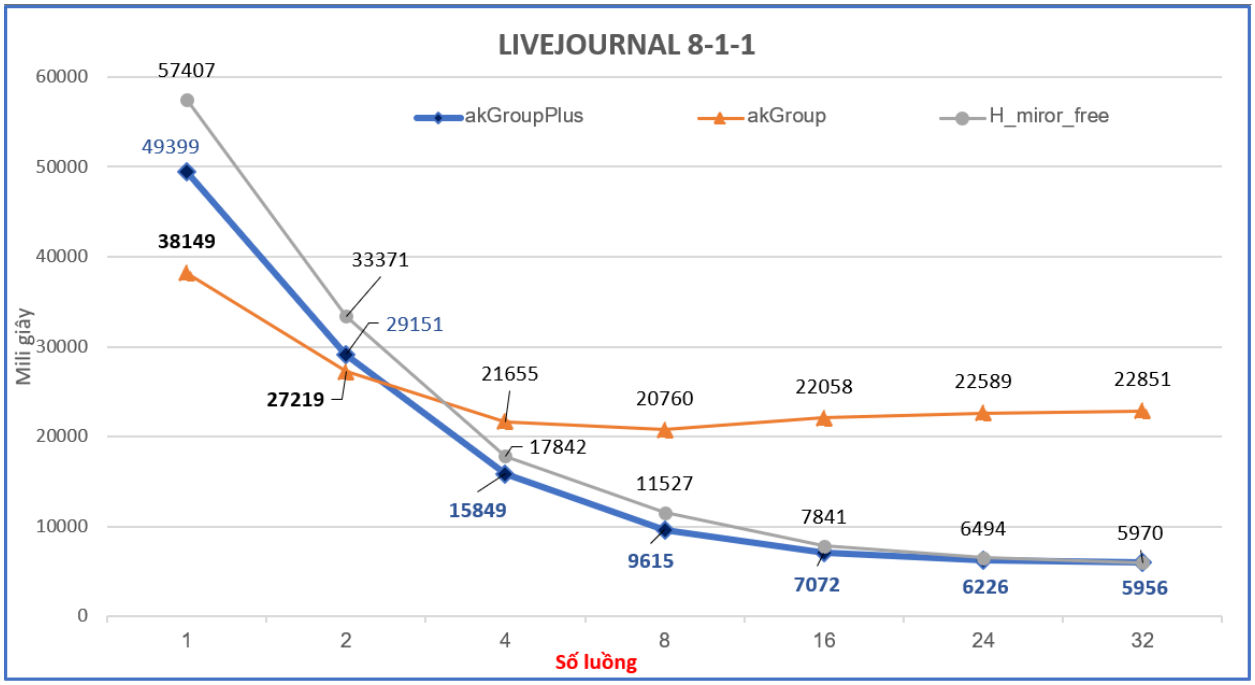


(a) Kết quả đánh giá với tập 8-1-1

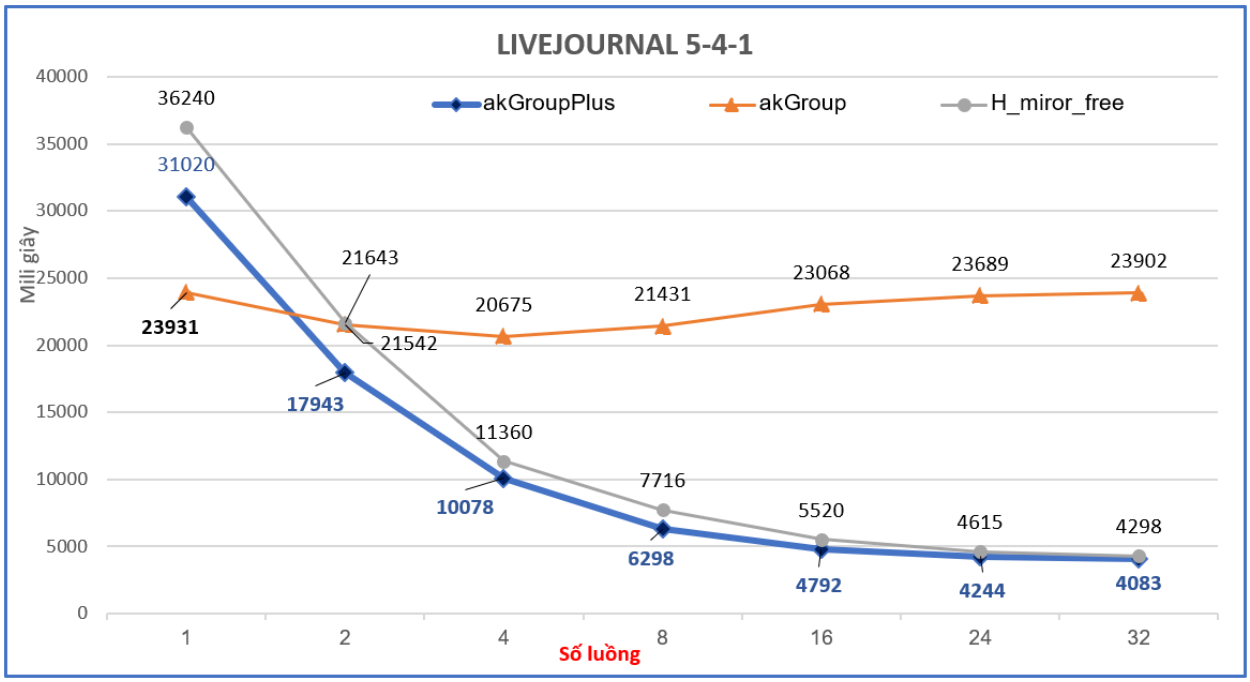


(b) Kết quả đánh giá với tập 5-4-1

Hình 3.9: Kết quả đánh giá với bộ dữ liệu Pokec Dataset



(a) Kết quả đánh giá với tập 8-1-1



(b) Kết quả đánh giá với tập 5-4-1

Hình 3.10: Kết quả đánh giá với bộ dữ liệu LiveJournal Dataset

Trong hình 3.8, thời gian thi hành của *akGroup* là lâu nhất và có xu hướng tăng khi số lượng luồng thi hành đồng thời được tăng lên. Với giải pháp này, do sử dụng tổ chức dữ liệu kiểu mảng một chiều, và di chuyển toàn bộ danh sách liền kề của một đỉnh về cuối với phép toán thêm cạnh. Do đó sẽ dẫn đến tình trạng mất đi tính cục bộ dữ liệu. Từ đó, khi tăng số luồng lên, thời gian thi hành của akGroup không những không giảm mà lại tăng. Lý do

chủ yếu đến từ việc mất tính cục bộ dữ liệu làm tăng tỷ lệ cache miss, và dẫn đến tăng thời gian thi hành.

Trong khi đó, *akGroupPlus* và *H_minor_free* có thời gian thi hành giảm tỉ lệ với số luồng tăng lên. Trong thực nghiệm này, akGroupPlus không có được hiệu năng tốt nhất khi thi hành với 1 luồng (tức thi hành tuần tự). Tuy nhiên, với các trường hợp còn lại, giải pháp này đều có thời gian thi hành là ít nhất so với hai giải pháp còn lại. Đối với bộ workload SigMod 5-4-1 với số lượng truy vấn cập nhật lớn (tức việc truy xuất đến dữ liệu đồ thị nhiều hơn), khi tăng từ 24 lên 32 luồng thì thời gian thi hành workload này không giảm mà lại tăng lên. Lý do việc thi hành 32 luồng song song có thể dẫn đến làm giảm tỷ lệ cache hit, tăng số lượng truy xuất dữ liệu đồ thị trong bộ nhớ chính và từ đó làm tăng thời gian thi hành.

Đối với bộ dữ liệu Pokec, như minh hoạ ở hình 3.9, thời gian thi hành của cả ba giải pháp đều giảm khi số lượng luồng thi hành đồng thời tăng lên, đặc biệt trong giai đoạn tăng từ 2 đến 16 luồng. Với tập dữ liệu kiểm tra Pokec 8-1-1, *akGroup* lại có kết quả tốt nhất với trường hợp chỉ sử dụng 1-2 luồng đồng thời. Tuy nhiên, khi số lượng luồng thi hành tăng lên, *akGroupPlus* lại là giải pháp luôn mang lại thời gian thi hành ít nhất. Cũng tương tự với bộ dữ liệu Pokec 5-4-1, akGroup có thời gian thi hành tốt nhất với trường hợp 1 luồng nhưng với các trường hợp còn lại *akGroupPlus* đều vượt trội so với akGroup và H_minor_free.

Đối với bộ dữ liệu LiveJournal, cũng tương tự như so với bộ dữ liệu Pokec, *akGroupPlus* luôn mang lại kết quả thời gian thi hành tốt hơn so với hai giải pháp còn lại ngoại trừ khi thi hành tuần tự.

Trong tất cả các thực nghiệm nêu trên, có thể thấy thời gian thi hành của cả ba giải pháp đều không giảm tuyến tính khi tăng tuyến tính số luồng đồng thời. Lý do chính của điều đó đầu tiên phải kể đến là các phép toán tương tranh bao hàm cả các phép cập nhật đồ thị nên có sự phụ thuộc lẫn nhau. Thứ hai, việc tăng số luồng lên cũng làm giảm tỷ lệ *cache hit* khi truy xuất dữ liệu đồ thị (do càng có nhiều luồng thi hành thì nguy cơ dữ liệu nằm trong cache bị thay thế bởi các luồng khác càng cao). Chính vì thế, với các đồ thị có dữ liệu quy mô lớn, số lượng luồng thi hành song song cần phải được lựa chọn theo những thực nghiệm cụ thể của từng bài toán.

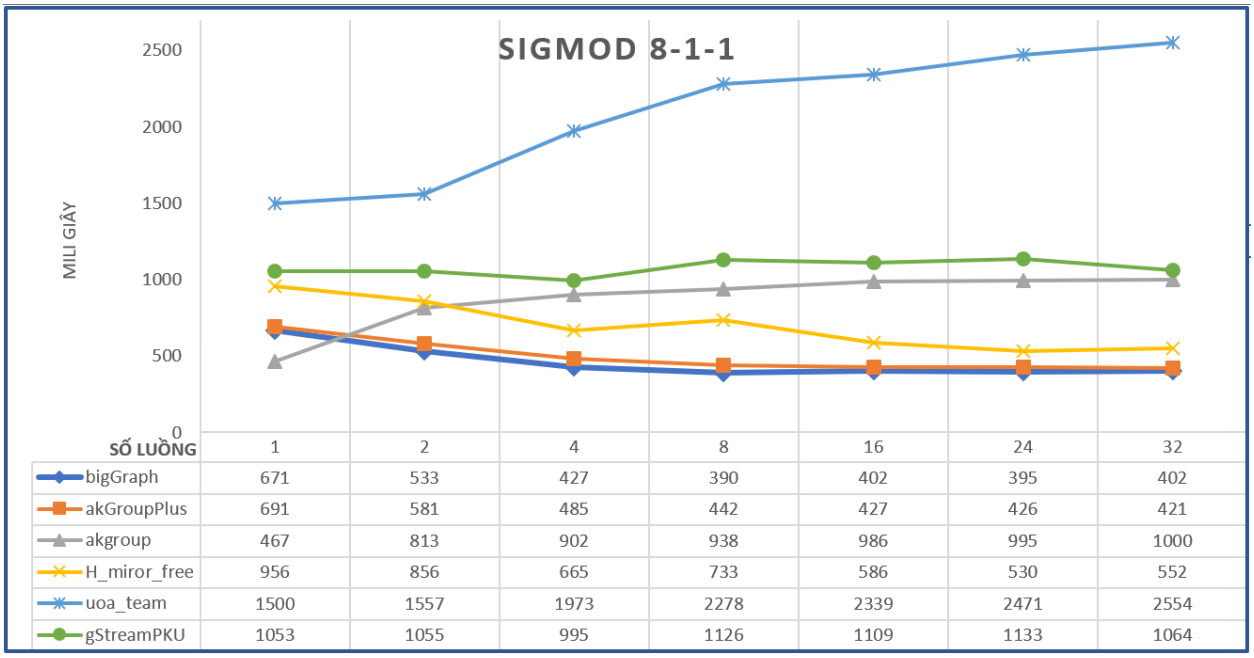
Giải pháp này của chúng tôi cũng đã được công bố tại hội thảo quốc tế ICCCI năm 2017 [DPH2]. Toàn bộ dữ liệu, kết quả thực nghiệm cũng như mã nguồn chương trình của giải pháp này có thể truy cập miễn phí tại địa chỉ `https://github.com/hanhdp/akGroupPlus/`.

3.6.3.4 Đánh giá giải pháp bigGraph

Với giải pháp **bigGraph**, chúng tôi đã đề xuất được cơ chế xử lý song song các phép toán cập nhật đồ thị. Trong thực nghiệm đánh giá **bigGraph**, chúng tôi đã tiến hành đánh giá so sánh **bigGraph** với cả **akGroup**; **H_minor_free**; và một số công cụ khác như **uoa_team** (đội giành giải nhì); **gStreamPKU**; và **while1** (hai đội đồng giải Ba cùng với

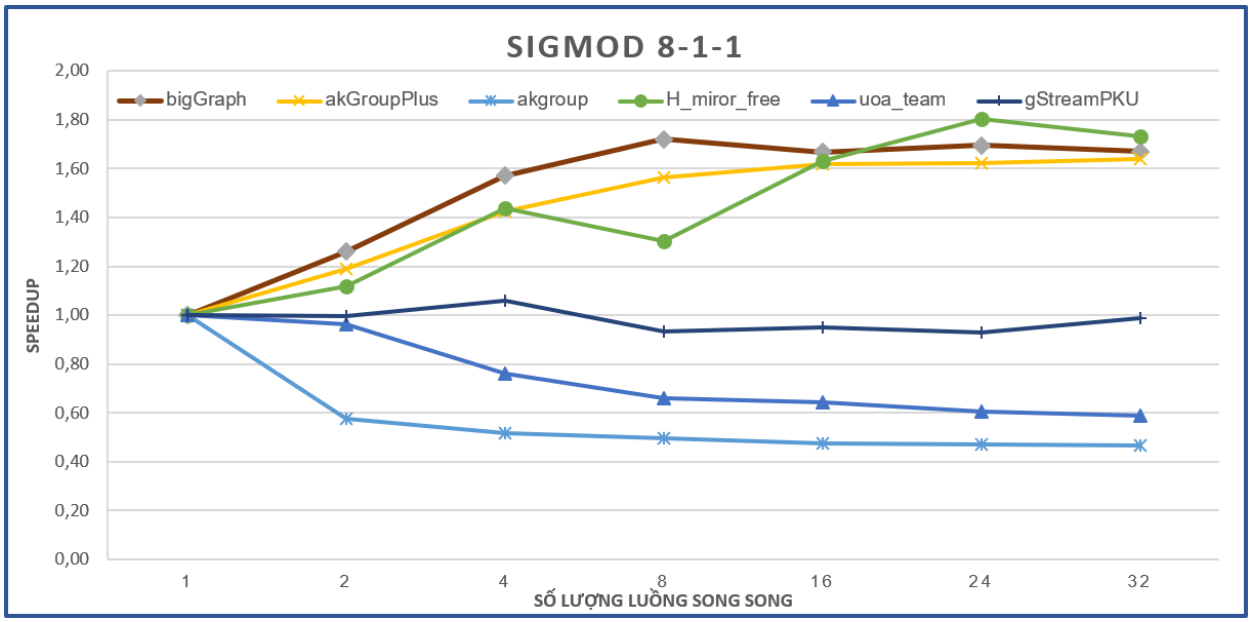
akGroup) dựa trên cả ba bộ dữ liệu mà chúng tôi đã đề cập ở trên. Với hệ thống tính toán hiệu năng cao của chúng tôi, số lượng luồng sẽ được thử nghiệm lần lượt là 1, 2, 4, 8, 16, 24, và 32 threads. Các kết quả thu được là thời gian thi hành các giải pháp được tính trung bình từ 10 lần chạy khác nhau. Một điều khá ngạc nhiên trong quá trình thử nghiệm các giải pháp là trong khi cả 5 giải pháp khác đều có kết quả chính xác, chỉ riêng giải pháp *while1* lại trả về kết quả không đúng với mong đợi, từ đó chúng tôi không đưa giải pháp này vào phần kết quả nữa. Các giải pháp như sử dụng $SNAP$ C++ hay $NetworkX$ (Python) cũng không được quan tâm trong thực nghiệm này do chúng chưa được cài đặt các giải pháp song song hoá truy vấn [DPH1].

Kết quả thực nghiệm chúng tôi đã thu được sẽ được thể hiện trên các hình dưới đây. Trong các hình kết quả đó, chúng tôi cũng hiển thị thêm thông tin về thời gian thi hành tốt nhất của từng giải pháp với cả 10 lần chạy và số lượng luồng thay đổi từ 1 đến 32 như đã nói trên.



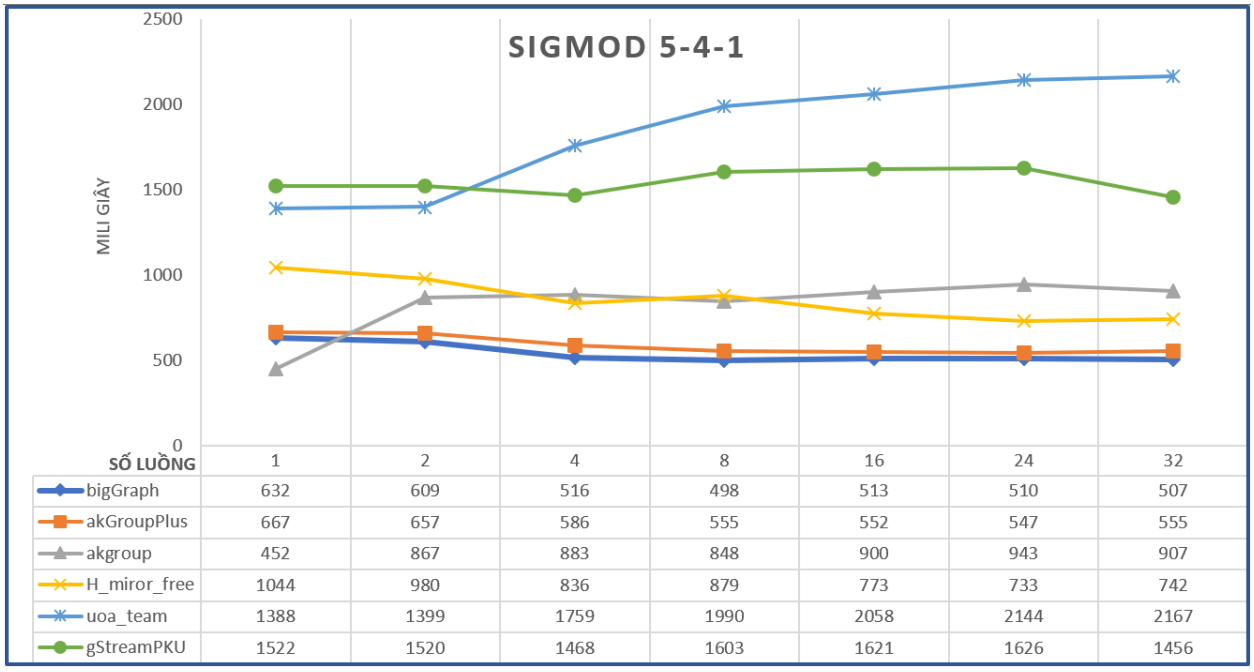
SỐ LUỒNG	1	2	4	8	16	24	32
bigGraph	671	533	427	390	402	395	402
akGroupPlus	691	581	485	442	427	426	421
akgroup	467	813	902	938	986	995	1000
H_miror_free	956	856	665	733	586	530	552
uoa_team	1500	1557	1973	2278	2339	2471	2554
gStreamPKU	1053	1055	995	1126	1109	1133	1064

(a) Thời gian thi hành



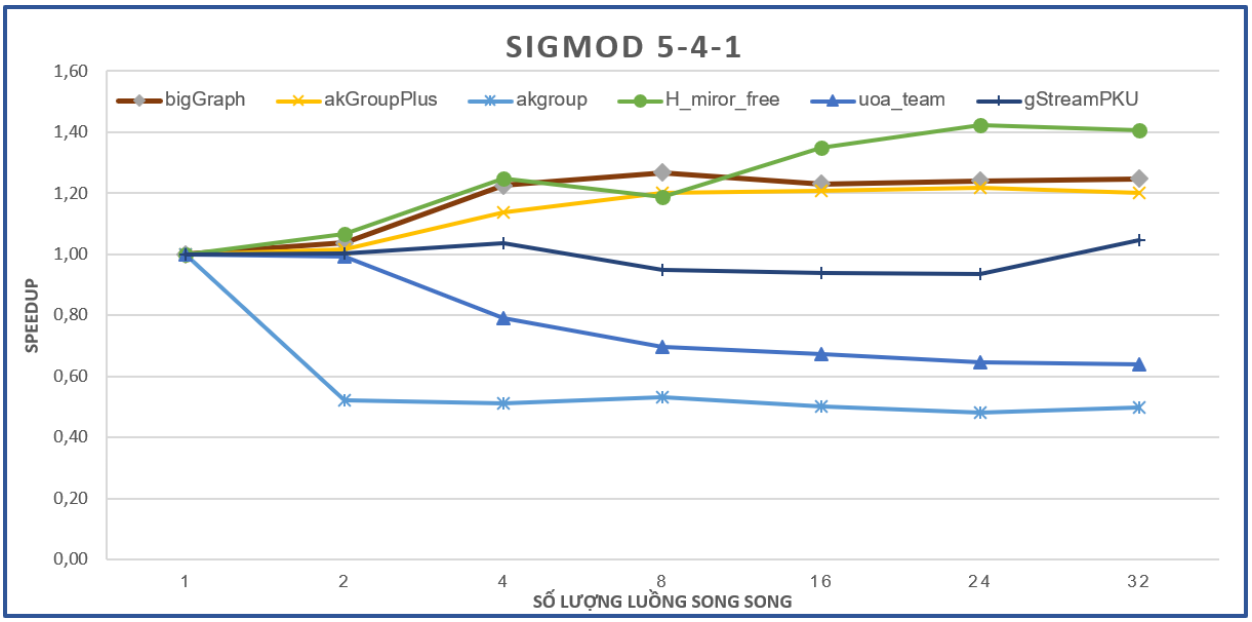
(b) Hệ số tăng tốc

Hình 3.11: Kết quả thực nghiệm với bộ dữ liệu SigMod 8-1-1



SỐ LUỒNG	1	2	4	8	16	24	32
bigGraph	632	609	516	498	513	510	507
akGroupPlus	667	657	586	555	552	547	555
akgroup	452	867	883	848	900	943	907
H_miror_free	1044	980	836	879	773	733	742
uoa_team	1388	1399	1759	1990	2058	2144	2167
gStreamPKU	1522	1520	1468	1603	1621	1626	1456

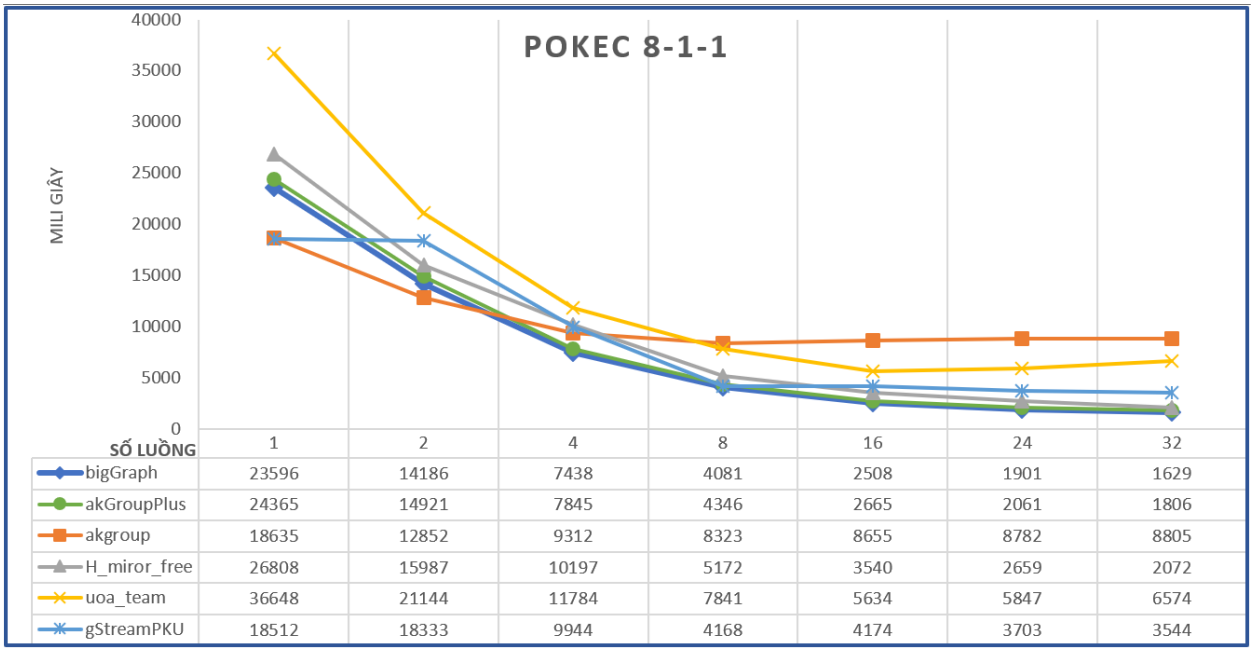
(a) Thời gian thi hành



(b) Hệ số tăng tốc

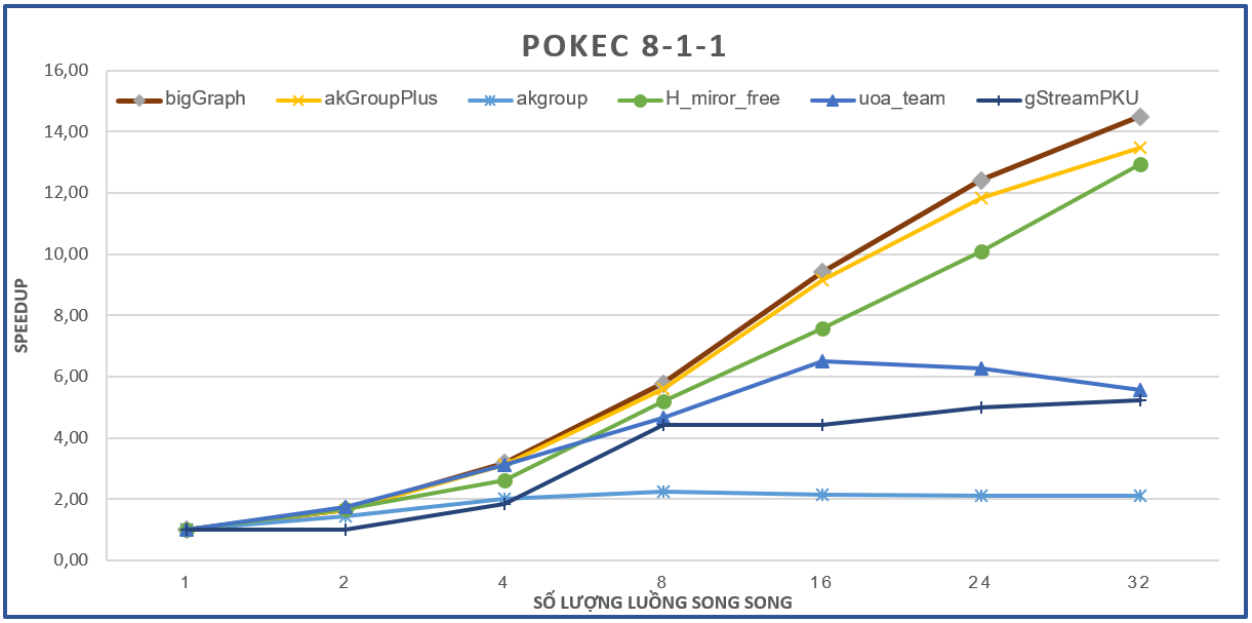
Hình 3.12: Kết quả thực nghiệm với bộ dữ liệu Sigmod 5-4-1

Với các bộ workloads SigMod 8-1-1 và SigMod 5-4-1, có thể thấy giải pháp **bigGraph** cho kết quả vượt trội so với các giải pháp khác, kể cả **akGroupPlus** và **H_minor_free**. Hai đồ thị biểu thị hệ số tăng tốc thu được cũng minh chứng khả năng song song hoá của **bigGraph** tuyến tính với số luồng thành đồng thời.



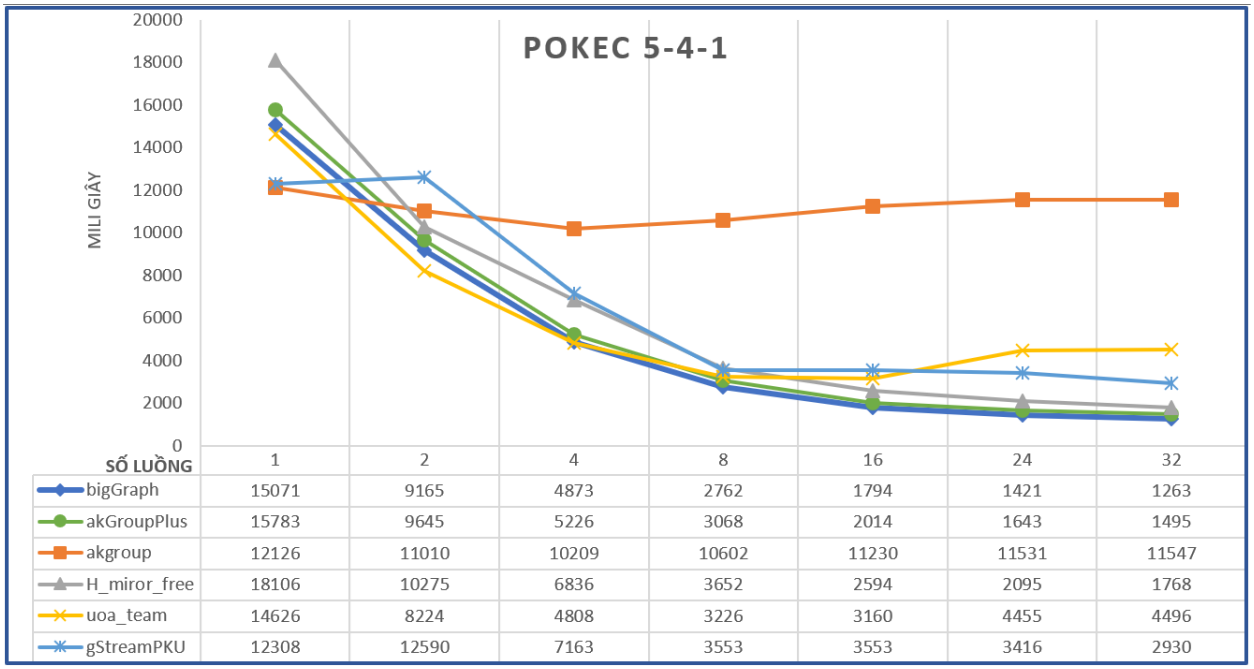
SỐ LUỒNG	1	2	4	8	16	24	32
bigGraph	23596	14186	7438	4081	2508	1901	1629
akGroupPlus	24365	14921	7845	4346	2665	2061	1806
akgroup	18635	12852	9312	8323	8655	8782	8805
H_miror_free	26808	15987	10197	5172	3540	2659	2072
uoa_team	36648	21144	11784	7841	5634	5847	6574
gStreamPKU	18512	18333	9944	4168	4174	3703	3544

(a) Thời gian thi hành



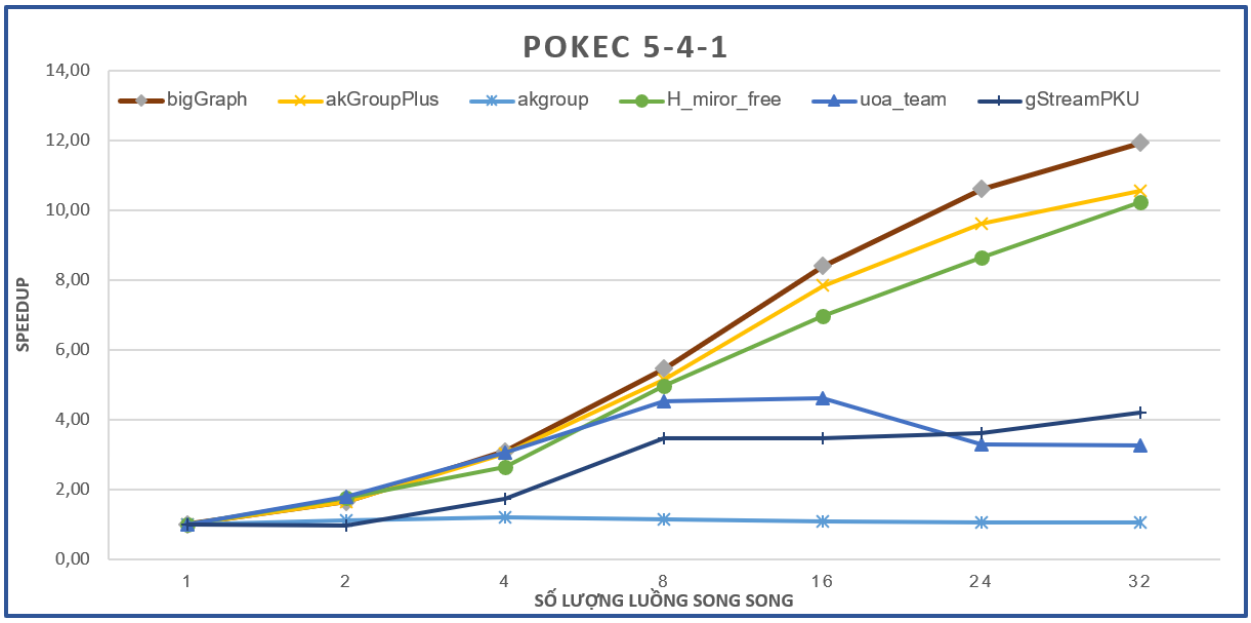
(b) Hệ số tăng tốc

Hình 3.13: Kết quả thực nghiệm với bộ dữ liệu Pokec 8-1-1



SỐ LUỒNG	1	2	4	8	16	24	32
bigGraph	15071	9165	4873	2762	1794	1421	1263
akGroupPlus	15783	9645	5226	3068	2014	1643	1495
akgroup	12126	11010	10209	10602	11230	11531	11547
H_miror_free	18106	10275	6836	3652	2594	2095	1768
uoa_team	14626	8224	4808	3226	3160	4455	4496
gStreamPKU	12308	12590	7163	3553	3553	3416	2930

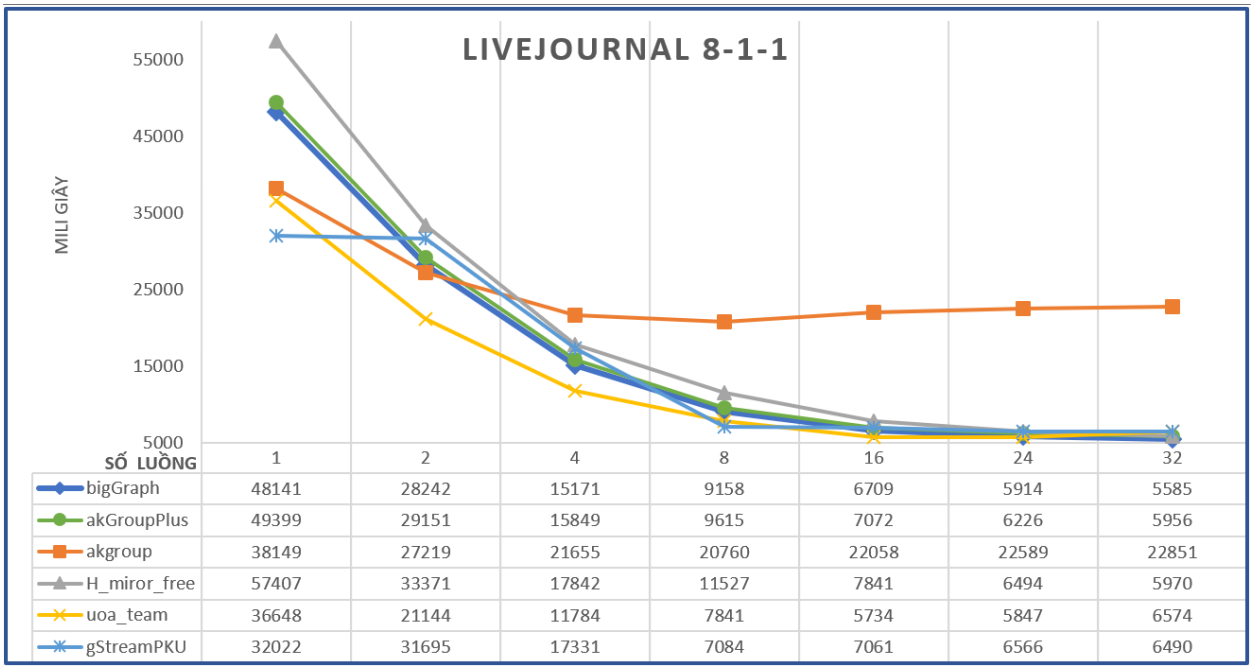
(a) Thời gian thi hành



(b) Hệ số tăng tốc

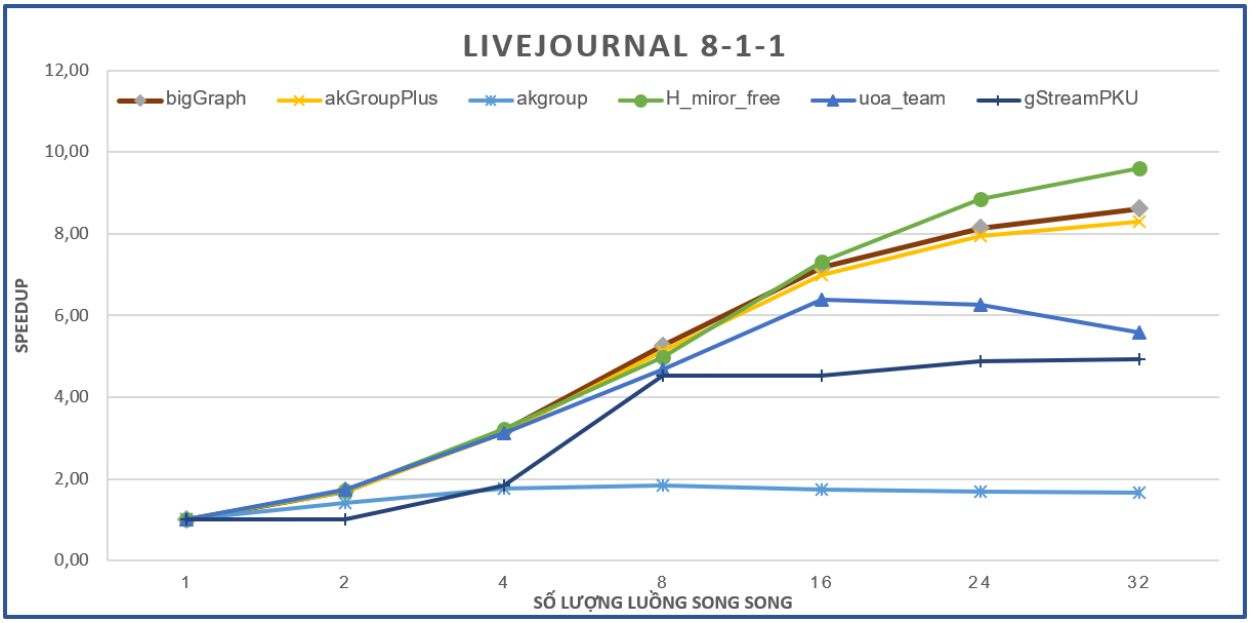
Hình 3.14: Kết quả thực nghiệm với bộ dữ liệu Pokec 5-4-1

Tương tự với các bộ workloads Pokec 8-1-1 và Pokec 5-4-1, thời gian thi hành tập truy vấn đều có xu hướng giảm khi số lượng luồng đồng thời tăng lên và **bigGraph** cũng cho kết quả vượt trội so với các giải pháp khác. Đối với các bộ dữ liệu này, do Pokec là đồ thị mạng xã hội có quy mô vừa phải đối với hạ tầng tính toán được chúng tôi sử dụng, hệ số tăng tốc của bigGraph có thể thấy tăng tương đối tuyến tính với số luồng đồng thời. Điều này cũng minh chứng rõ hơn hiệu quả song song của giải thuật bigGraph so với các giải thuật còn lại.



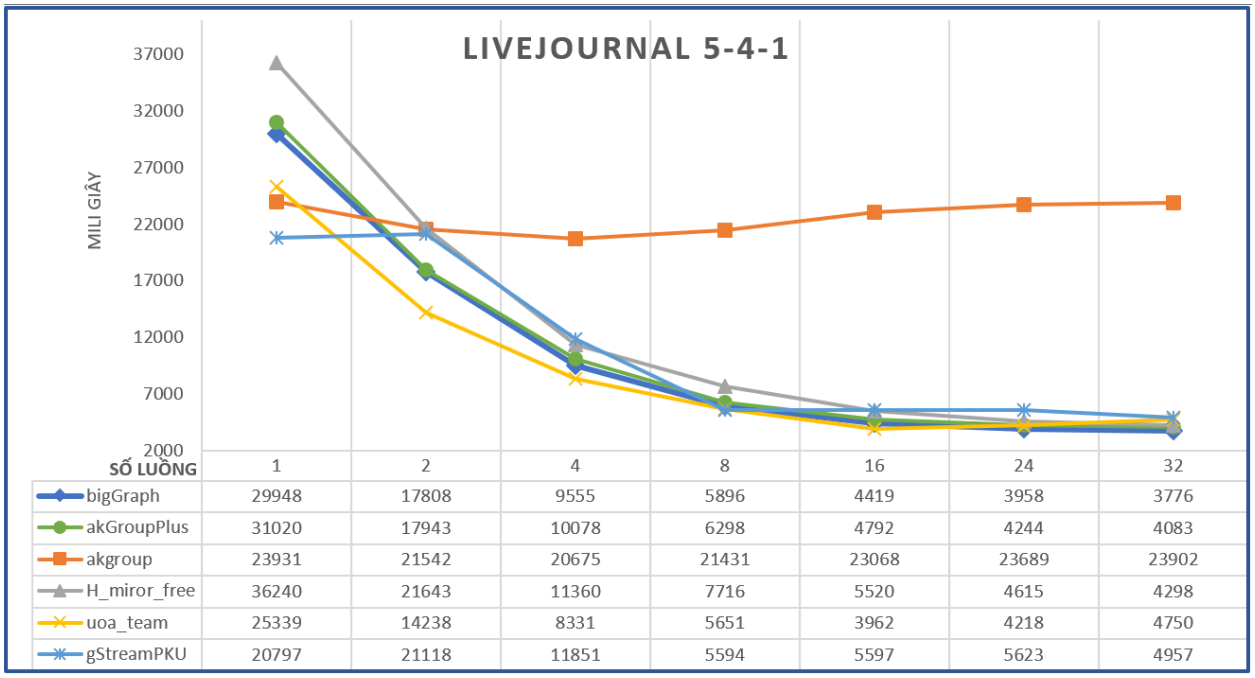
SỐ LUỒNG	1	2	4	8	16	24	32
bigGraph	48141	28242	15171	9158	6709	5914	5585
akGroupPlus	49399	29151	15849	9615	7072	6226	5956
akgroup	38149	27219	21655	20760	22058	22589	22851
H_miror_free	57407	33371	17842	11527	7841	6494	5970
uoa_team	36648	21144	11784	7841	5734	5847	6574
gStreamPKU	32022	31695	17331	7084	7061	6566	6490

(a) Thời gian thi hành



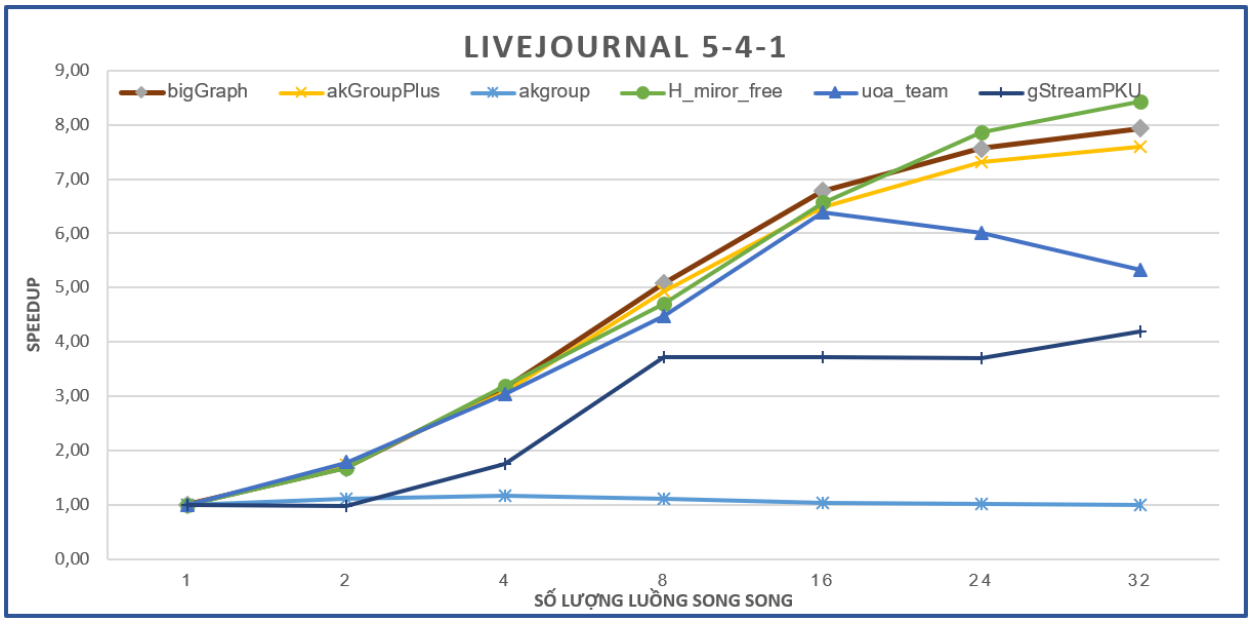
(b) Hệ số tăng tốc

Hình 3.15: Kết quả thực nghiệm với bộ dữ liệu LiveJournal 8-1-1



SỐ LUỒNG	1	2	4	8	16	24	32
bigGraph	29948	17808	9555	5896	4419	3958	3776
akGroupPlus	31020	17943	10078	6298	4792	4244	4083
akgroup	23931	21542	20675	21431	23068	23689	23902
H_miror_free	36240	21643	11360	7716	5520	4615	4298
uoa_team	25339	14238	8331	5651	3962	4218	4750
gStreamPKU	20797	21118	11851	5594	5597	5623	4957

(a) Thời gian thi hành



(b) Hệ số tăng tốc

Hình 3.16: Kết quả thực nghiệm với bộ dữ liệu LiveJournal 5-4-1

Kết quả thực nghiệm với hai bộ workloads LiveJournal 8-1-1 và LiveJournal 5-4-1 cũng đều minh chứng bigGraph cho thời gian thi hành tốt nhất so với các giải pháp khác; kể cả giải pháp akGroupPlus đã nêu trên. Như vậy có thể khẳng định, với cả ba bộ dữ liệu thử nghiệm và hai workloads 8-1-1 hay 5-1-1 thì bigGraph đều có hiệu năng thi hành tốt hơn so với các giải pháp còn lại; hệ số tăng tốc của bigGraph đều có xu thế tăng khi số lượng luồng song song được tăng lên.

Thời gian thi hành ngắn nhất của mỗi giải pháp trên mỗi bộ dữ liệu được thống kê trong

bảng dưới đây. Số luồng song song có thời gian thi hành ngắn nhất cũng được hiển thị kèm theo trong bảng này:

Bảng 3.6: Thống kê hiệu năng tốt nhất

Giải pháp	**SigMod**		**Pokec**		**LiveJournal**	
	8-1-1	5-4-1	8-1-1	5-4-1	8-1-1	5-4-1
akGroup	453 (1T)	**447** (1T)	8269 (8T)	10148 (4T)	20597 (8T)	20317 (2T)
H_minor_free	517 (24T)	725 (24T)	1987 (32T)	1690 (32T)	5878 (32T)	4219 (32T)
uoa_team	1427 (1T)	1359 (1T)	5385 (16T)	2857 (16T)	5587 (16T)	3814 (16T)
gSteamPKU	945 (4T)	1401 (32T)	3375 (32T)	2887 (32T)	6256 (32T)	4640 (32T)
akGroupPlus	410 (16T)	526 (8T)	1785 (32T)	1469 (32T)	5848 (32T)	4013 (32T)
bigGraph	**373** (8T)	463 (32T)	**1599** (32T)	**1242** (32T)	**5537** (32T)	**3718** (32T)

Từ bảng thống kê này có thể thấy bigGraph cho kết quả tốt nhất so với toàn bộ các giải pháp còn lại. Điều đó có được dựa trên phương pháp tổ chức dữ liệu đồ thị phù hợp nhúng trạng thái cạnh vào 2 bit cuối của mỗi đỉnh liền kề và kỹ thuật song song hoá cả đối với các phép toán cập nhật đồ thị lẫn truy vấn khoảng cách ngắn nhất.

Toàn bộ mã nguồn, dữ liệu và kết quả thử nghiệm của giải pháp này đều có thể truy cập miễn phí tại địa chỉ `https://github.com/hanhdp/bigGraph/`. Kết quả này cũng được chúng tôi công bố trong tạp chí quốc tế Transactions on Computational Collective Intelligence, Springer, năm 2018 [DPH3].

3.7 Kết chương 3

Trong chương này, chúng tôi đã đặc tả bài toán xử lý các truy vấn khoảng cách ngắn nhất trên đồ thị động, quy mô lớn. Về cơ bản, các phép toán trên đồ thị có thể được chia thành hai lớp: (i) các phép toán cập nhật bao gồm thêm, xoá cạnh (ngay cả đối với đồ thị thuộc tính, việc thay đổi thuộc tính nào đó trong đồ thị cũng có thể quy về phép toán xoá cạnh rồi bổ sung thêm cạnh có thuộc tính mới chẳng hạn); và (ii) các phép toán truy vấn trên đồ thị, được tiến hành thông qua các giải thuật duyệt đồ thị hoặc xác định đường đi ngắn nhất giữa các đỉnh chẳng hạn. Khi số lượng đỉnh cũng như cạnh trên đồ thị lớn lên, việc tiến hành xử lý các truy vấn đó rõ ràng cần phải có những phương pháp, kỹ thuật phù hợp để có thể hiệu năng xử lý truy vấn tốt nhất có thể.

Với việc tập trung nghiên cứu các đồ thị không trọng số, có hướng và có quy mô đỉnh/cạnh lớn, trong bài toán xử lý các phép toán tương tranh chúng tôi chú trọng đến trường hợp các phép toán cập nhật và truy vấn khoảng cách ngắn nhất giữa các đỉnh được gửi đến liên tục, đồng thời trên đồ thị. Lúc đó, các phép toán đó cần phải được thi hành với điều kiện đảm bảo được tính toàn vẹn, nhất quá của những dữ liệu đồ thị. Từ đó, các phép toán tương

tranh trong đồ thị được mô hình hoá như một lịch thi hành các phép toán S trên đồ thị G.

Từ mô hình bài toán đó, việc tiến hành thực thi lịch S được chúng tôi đề xuất thông qua ba giải pháp khác nhau. Trong giải pháp 1 (akGroup), để khai thác các hệ thống tính toán có dung lượng bộ nhớ chính lớn và các bộ vi xử lý nhiều lõi, chúng tôi đã sử dụng ba ý tưởng chính chính để giải quyết bài toán đặt ra: tổ chức dữ liệu đồ thị thích hợp để tăng tốc độ truy cập trong bộ đệm CPU; giảm thiểu không gian tìm kiếm bằng cách lựa hướng duyệt có số lượng con và cháu ít hơn đối với thuật toán duyệt hai chiều bBFS; và thi hành các truy vấn khoảng cách ngắn nhất một cách hiệu quả thông qua phương pháp xử lý song song dựa vào bộ thư viện CilkPlus. Ở giải pháp này, dữ liệu đồ thị được tổ chức trên hai mảng chứa toàn bộ các đỉnh liền kề và hai mảng chỉ mục để lưu số đỉnh liền kề và vị trí bắt đầu trong mảng dữ liệu đồ thị. Cách tổ chức này được tiến hành đối với cả chiều đi (outgoing) và chiều đến (incoming) để cho phép có thể tiến hành tính khoảng cách ngắn nhất dựa theo duyệt BFS cả hai chiều. Mặc dù cách thức tổ chức dữ liệu này cho phép nâng cao hiệu quả ban đầu của dữ liệu đồ thị (dữ liệu liền kề nên có tính địa phương cao, từ đó nâng cao được tỷ lệ cache hit), nhưng các phép toán cập nhật (đặc biệt thao tác thêm cạnh) lại có hiệu năng không cao.

Từ các điểm hạn chế ở giải pháp 1, chúng tôi đã đề xuất giải pháp 2 với mô hình tổ chức dữ liệu đồ thị khác đi. Với phương pháp nhúng kèm trạng thái cạnh đồ thị (sử dụng 2 bits cuối của định danh đỉnh liền kề), các phép toán cập nhật được chia làm hai giai đoạn: chuyển trạng thái các cạnh có thao tác cập nhật (thêm/xoá) thành UNKNOWN và sau khi hoàn thành các truy vấn tính khoảng cách xong mới ghi nhận thực sự các cạnh đó với các trạng thái thêm (ALIVE) hay bị xoá đi (DEAD). Với cách tổ chức dữ liệu này, việc thi hành toàn bộ các truy vấn khoảng cách trong S được tập hợp lại và xử lý song song trên hệ thống tính toán. Từ đó, hiệu năng của quá trình xử lý S đã được cải thiện như đã được minh chứng trong thực nghiệm thứ 2 của chúng tôi. Mặc dù vậy, mới chỉ có các truy vấn khoảng cách trong S được tiến hành song song trong khi các phép toán cập nhật vẫn chỉ được xử lý tuần tự.

Trong giải pháp 3, chúng tôi đã đề xuất kỹ thuật cho phép tiến hành song song các phép toán cập nhật với ý tưởng có thể tiến hành song song các phép toán cập nhật trên các danh sách đỉnh liền kề khác nhau. Từ ý tưởng đó, chúng tôi đã xây dựng phương án tiến hành song song các phép toán cập nhật. Các thực nghiệm được chúng tôi tiến hành với giải pháp này cũng đã minh chứng được hiệu quả của quá trình song song cả các phép toán cập nhật lẫn tính khoảng cách ngắn nhất.

Với các giải pháp được đề xuất trong luận án, các giải thuật song song đều đảm bảo tính cục bộ thi hành trong mỗi luồng. Do đó, không có sự trao đổi giữa các luồng và không cần phải có cơ chế kiểm soát tương tranh.

Từ đó, với các truy vấn đồng thời trên đồ thị có hướng, không trọng số, quy mô lớn, rõ ràng giải pháp bigGraph là giải pháp tối ưu nhất so với các giải pháp còn lại (kể cả akGroup và akGroupPlus). Do đó, bigGraph là phù hợp nhất để giải quyết bài toán nghiên cứu đã

đặt trong chương này của Luận án.

Trong thực tế, ngoài việc cần phải tiến hành các truy vấn khoảng cách ngắn nhất trên đồ thị động như trong bài toán đã đặc tả trong chương này, việc phân tích đồ thị còn cần phải tính toán xác định một số độ đo như đã giới thiệu trong Chương 2 (chẳng hạn như độ trung tâm gần, độ trung tâm trung gian, ...). Vì thế, dựa trên cách tiếp cận tổ chức dữ liệu phù hợp và tính toán song song đã minh chứng được hiệu quả trong bài toán nêu trên, chúng tôi đã tiến hành nghiên cứu và đề xuất được một số những giải pháp nâng cao được hiệu quả trong quá trình tính toán một số độ đo trung tâm phục vụ phân tích đồ thị quy mô lớn. Các kết quả liên quan đến vấn đề đó sẽ được trình bày chi tiết trong Chương sau.

Chương 4

NÂNG CAO HIỆU NĂNG TÍNH ĐỘ TRUNG TÂM TRÊN ĐỒ THỊ

4.1 Giới thiệu

Lý thuyết đồ thị hiện được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau, từ các bài toán trong giao vận, tin sinh học, quy hoạch mạng, phân tích mạng Internet, mạng xã hội... Để phân tích các đồ thị, độ trung tâm (centrality) chính là độ đo quan trọng và được sử dụng rộng rãi hiện nay[2], hướng đến việc tìm các đỉnh *"quan trọng"* nhất, hay gây ảnh hưởng lớn nhất tới các đỉnh khác trong đồ thị. Khi áp dụng khái niệm này cho các lĩnh vực khác nhau, chúng ta có thể tìm được các đỉnh chính trong mạng Internet, hay các đỉnh làm lan truyền dịch bệnh khi mô hình hóa bài toán lan bệnh dịch bằng đồ thị. Thực tế, khái niệm "quan trọng" được định nghĩa theo nhiều cách khác nhau khi phân tích đồ thị. Từ đó, cũng có nhiều độ đo trung tâm được đề xuất để làm rõ được tính "quan trọng" khi phân tích mạng đó.

Một trong những ứng dụng điển hình của lý thuyết đồ thị hiện nay chính là trong mô hình hoá các mạng xã hội [53, 55, 106]. Để quản lý mạng xã hội, nhiều phương pháp phân tích mạng xã hội đã được đề xuất và sử dụng trong thực tế. Phân tích mạng xã hội được định nghĩa là quá trình điều tra các cấu trúc mạng xã hội thông qua việc sử dụng mạng và lý thuyết đồ thị [85]. Vì vậy, phương pháp này được coi là một kỹ thuật quan trọng trong xã hội học hiện đại.

Trong chương này, chúng tôi sẽ đề cập đến một số phương pháp hiện đang được sử dụng để tính một số độ đo trung tâm, cụ thể là hai độ đo phổ dụng: (i) độ trung tâm gần và (ii) độ trung tâm trung gian trên đồ thị không trọng số, có thể có hướng hoặc vô hướng. Trên cơ sở đó, luận án sẽ trình bày phương pháp cải thiện hiệu năng tính toán hai độ đo trung tâm này và tiến hành thử nghiệm trên các bộ dữ liệu mạng xã hội do một số tổ chức có uy tín công bố. Một phần đóng góp của chương này được thể hiện trong công bố sau:

- [**DPH4**] **Phuong-Hanh DU**, Hai-Chau NGUYEN, Kim-Khoa NGUYEN, and Ngoc-

Hoa NGUYEN."An Efficient Parallel Algorithm for Computing the Closeness Centrality in Social Networks". *In The Ninth International Symposium on Information and Communication Technology (SoICT 2018)*, pp. 456-462, December, 2018. ACM, USA. (SCOPUS, WoS)

4.2 Bài toán đặt ra

Như đã trình bày phần trước về các độ đo trung tâm, hai trong những chỉ dấu trung tâm được sử dụng rộng rãi nhất trong các bài toán phân tích đồ thị là *"độ trung tâm gần" (closeness centrality)* và *"độ trung tâm trung gian" (betweenness centrality)* [53]. Trong các phép toán phân tích đồ thị, độ trung tâm gần được xem như khái niệm *căn bản* của độ đo trung tâm [52]. Việc tính độ trung tâm gần của một đỉnh trong đồ thị đòi hỏi phải giải quyết bài toán tìm tất cả các cặp đường đi ngắn nhất giữa các đỉnh (tức phải duyệt SSSP với tất cả các đỉnh). Vì vậy cần phải thực hiện thuật toán tìm kiếm theo chiều rộng trước (BFS) trên một mạng không có trọng số hoặc thuật toán Dijkstra trên một mạng có trọng số. Tuy nhiên, khi số lượng thành viên (tức số đỉnh) cũng như số lượng quan hệ (số cạnh) rất lớn, chẳng hạn như mạng xã hội Facebook, thì cần phải có những phương pháp theo kiểu "chia để trị" mới có thể tính được độ đo này [86].

Một điểm cũng cần phải nhấn mạnh thêm ở đây là khi nghiên cứu cải thiện hiệu năng tính độ trung tâm gần và độ trung tâm trung gian, chúng tôi chỉ quan tâm đến những đồ thị không trọng số, có thể có hướng hoặc vô hướng.

4.2.1 Tính độ trung tâm gần

Quá trình tính độ trung tâm gần của các đỉnh trong đồ thị $G = (V, E)$ có thể được minh hoạ bằng Giải thuật 4.1. Giải thuật này sử dụng phương pháp duyệt đồ thị theo chiều rộng

trước (BFS) đối với mỗi đỉnh v của V và lưu lại giá trị trong vector $CC[v]$ [79].

Thuật toán 4.1: Giải thuật cơ bản tính độ trung tâm gần

Input: $G = (V, E); \Gamma_G(v)$ biểu diễn danh sách đỉnh liền kề của v
Output: $CC[.]$ for all $v \in V$

```
1  CC[v] ← 0, ∀v ∈ V ;
2  Sum[v] ← 0, ∀v ∈ V ;
3  foreach s ∈ V do
4      FC[s] ← 0;
5      Q ← empty queue;
6      Q.push(s);
7      dst[s] ← 0;
8      CC[s] ← 0;
9      dst[v] ← −1, ∀v ∈ V s;
10     while Q is not empty do
11         v ← Q.pop();
12         forall w ∈ Γ_G(v) do
13             if dis[w] = ∞ then
14                 Q.push(w);
15                 dst[w] ← dst[v] + 1;
16                 Sum[v] ← dst[w];
17             end
18         end
19     end
20     CC[s] ← 1/Sum[s] ;
21 end
22 return CC[.];
```

Độ phức tạp tính toán của giải thuật 4.1 này là $O(|V| * (|V| + |E|))$. Thực tế, việc thi hành tính toán xác định độ trung tâm gần của tất cả các đỉnh trong đồ thị có quy mô lớn cả về số đỉnh và số cạnh thường không thể thi hành tuần tự do độ phức tạp tính toán quá lớn [21]. Từ đó có thể thấy với những mạng xã hội như Facebook hay Youtube, thời gian thi hành tính độ trung tâm gần đối với tất cả đỉnh là rất lớn. Điều này cũng được thể hiện trong thực nghiệm của chúng tôi (được trình bày ở mục sau), chẳng hạn với một tập dữ liệu nhỏ thể hiện cộng đồng trong Youtube, việc tính độ trung tâm gần của 1.134.890 đỉnh với 2.987.624 cạnh đã mất 147.924 giây [DPH4].

Như đã phân tích trong một số nghiên cứu liên quan ở mục 1.2, song song hoá giải thuật 4.1 được xem như một trong những phương pháp hiệu quả nhất để có thể cải tiến hiệu năng tính toán độ trung tâm gần [19]. Tuy nhiên, các giải pháp như NetworKit[99] hay TeexGraph[101],..., chưa xét và khai thác đến mô hình phân cấp bộ nhớ trong các hệ thống

tính toán để giảm được tỷ lệ cache miss và tăng cache hit [DPH1]. Một số công trình nghiên cứu khác đã sử dụng các hệ thống tính toán hiệu năng cao phân tán để tính toán độ trung tâm gần, chẳng hạn như Pregel [69], GraphLab [107] hay Giraph [25] đã có thể thao tác với những mạng có đến hàng nghìn tỷ cạnh [24]. Tuy vậy, các bộ công cụ đó cần phải được triển khai trên những nền tảng tính toán phức tạp chẳng hạn như siêu máy tính [107] và không thực sự hiệu quả đối với bài toán cần tính toán độ trung tâm với các mạng thực cỡ không quá lớn như Facebook và môi trường tính toán hạn chế.

Từ những phân tích, đánh giá đó, trong luận án này, chúng tôi sẽ chú trọng nghiên cứu phương pháp, kỹ thuật để cải thiện quá trình tính độ trung tâm gần, đặc biệt đối với những đồ thị có số đỉnh/cạnh lớn (hàng triệu).

4.2.2 Tính độ trung tâm trung gian

Độ trung tâm trung gian BC, được Freeman đề xuất từ 1977 [36], là một trong những độ đo được sử dụng rộng rãi trong phân tích đồ thị nói chung nhằm xác định những đỉnh quan trọng trong đồ thị. Độ đo này đã được áp dụng để phân tích đồ thị/mạng trong nhiều lĩnh vực khác nhau như trong vận tải [112], sinh học - y tế [16], phân tích mạng xã hội để phát hiện cộng đồng [23], phát hiện nguy cơ khủng bố [53], ...

Từ định nghĩa độ trung tâm trung gian 2.22 đã được trình bày ở chương 2, để tính được độ đo này cho tất cả các đỉnh V trong đồ thị $G = (V, E)$, chúng ta cần phải giải bài toán tìm đường đi ngắn nhất trên tất cả các cặp đỉnh trong G (tức bài toán APSP). Rõ ràng, khi số đỉnh và số cạnh của G lớn, việc thi hành quá trình tính APSP sẽ có thời gian rất lớn nếu áp dụng phương pháp tính thông thường sử dụng giải thuật Floyd-Warshall (độ phức tạp tính toán là $O(|V|^3)$) hay giải thuật Johnson (với độ phức tạp $O(|V|^2 log(|V|) + |V||E|)$). Cho đến thời điểm này, giải thuật rất hiệu quả để tính chính xác độ trung tâm trung gian với tất cả các đỉnh trong G vẫn là giải thuật do Brandes đề xuất từ năm 2001 với độ phức tạp thời gian là $O(|V|.|E|)$ trên đồ thị không trọng số và $O(|V|.|E| + |V|^2.log(|E|))$ trên đồ thị có trọng số [105]. Mã giả của thuật toán Brandes [15] với đồ thị không trọng số được

minh hoạ như trong giải thuật 4.2 dưới đây:

Thuật toán 4.2: Giải thuật cơ bản tính độ trung tâm trung gian

Input: $G = (V, E)$, được tổ chức như mảng vector $Edges[][]$
Data: queue $Q \leftarrow empty$, stack S khởi tạo rỗng và có thể chứa được $|V|$ đỉnh ;
$dist[v]$: lưu khoảng cách từ đỉnh nguồn đến v ;
$Pred[v]$: chứa danh sách các đỉnh trên đường đi ngắn nhất từ đỉnh nguồn đến v ;
$\sigma[v]$: số đường đi ngắn nhất từ đỉnh nguồn đến v ;
$\delta[v]$: độ thuộc của đỉnh nguồn vào đỉnh v ;
Output: $BC[.]$ với mọi $v \in V$

```
1  foreach s ∈ V do
       /* Pha 1. Duyệt theo kiểu bài toán SSSP                          */
2      foreach v ∈ V do Pred[v] ← empty list ;
3      foreach v ∈ V do dist[v] ← ∞; σ[v] ← 0 ;
4      dist[s] ← 0; σ[s] ← 1; Q.push(s) ;
5      while Q not empty do
6          v ← Q.pop(); S.push(v) ;
7          foreach w ∈ Edges[v] do
               /* w chưa được duyệt                                     */
8              if dist[w] == ∞ then dist[w] ← dist[v] + 1; Q.push(w) ;
               /* (v,w) nằm trên đường đi ngắn nhất                     */
9              if dist[w] == dist[v] + 1 then σ[w] ← σ[w] + σ[v]; Pred[w].push_back(v);
10         end
11     end
       /* Pha 2: Tích luỹ - tính ngược lại độ thuộc từ các đỉnh đã duyệt */
12     foreach v ∈ V do δ[v] ← 0;
13     while S not empty do
14         w ← S.pop() ;
15         for v ∈ Pred[w] do δ[v] ← δ[v] + (σ[v]/σ[w]).(1 + δ[w]);
16         if w ≠ s then BC[w] ← BC[w] + δ[w];
17     end
18 end
19 return BC[.] ;
```

Trong giải thuật này, việc tính độ trung tâm trung gian được lặp lại với tất cả các đỉnh $s \in V$, mỗi lần tính $\delta(s|v)$ với mọi $v \in V$ trong hai pha. Pha đầu tiên là pha duyệt theo chiều rộng trước BFS để xác định khoảng cách và đường đi ngắn nhất từ s đến các đỉnh còn lại trong V. Pha thứ hai duyệt ngược lại các đỉnh đã duyệt ở pha 1, tức tương ứng với từ

đỉnh xa s nhất quay về, để tính độ phụ thuộc theo phương trình (4.1) dưới đây [15]:

$$\delta(s|v) = \sum_{w:(v,w)\in E \ and \ dist(s,w)=dist(s,v)+1} \frac{\delta(s,v)}{\delta(s,w)}.(1+\delta(s,w)) \tag{4.1}$$

Tương tự như đối với độ trung tâm gần, ngay cả với giải thuật Brandes này, quá trình tính độ trung tâm trung gian này cũng mất rất nhiều thời gian nếu số đỉnh/cạnh của đồ thị quá lớn. Chẳng hạn với 100 triệu đỉnh và 100 triệu cạnh thì tổ hợp số phép tính cần phải thi hành khi tính độ trung tâm đã là rất lớn: 10^{16} hay 10 triệu tỷ.

Để cải thiện tốc độ tính độ trung tâm trung gian, nhiều giải pháp tính xấp xỉ độ đo này đã được đề xuất, chẳng hạn sử dụng kỹ thuật lấy mẫu [20, 32, 89, 68]; sử dụng mẹo để cải thiện hiệu năng [6, 12, 88, 91, 105]. Cách tiếp cận tính toán song song trên hệ thống tính toán đa chíp, đa lõi cũng được áp dụng trong việc cải thiện hiệu năng tính độ trung tâm trung gian BC(chẳng hạn như NetworKit [99], TeexGraph [101]. Nâng cao hiệu năng tính BC cũng được quan tâm nghiên cứu trên GPU, chẳng hạn như giải pháp MGBC [13]. Cũng tương tự như đối với việc tính độ trung tâm gần CC, trên các siêu máy tính cũng có những giải pháp như GraphLab [107] hay Apache Giraph [25].

Trong khuôn khổ luận án này, chúng tôi chỉ tập trung nghiên cứu việc cải thiện, nâng cao hiệu năng tính chính xác độ trung tâm trung gian BC bằng cách tổ chức cấu trúc dữ liệu đồ thị một cách phù hợp và xây dựng kỹ thuật tính toán song song giải thuật Brandes theo mô hình bộ nhớ chia sẻ (việc kết hợp thêm với GPU sẽ được định hướng nghiên cứu trong thời gian tới).

4.3 Nâng cao hiệu năng tính độ trung tâm

Dựa vào những phân tích đã nêu, chúng tôi đã tiến hành nghiên cứu và xây dựng giải pháp nâng cao hiệu năng tính độ đo trung tâm gần trên đồ thị không trọng số. Giải pháp của chúng tôi dựa trên tư tưởng:

1. Sử dụng cấu trúc dữ liệu phù hợp để nâng cao tính cục bộ dữ liệu, từ đó nâng cao tỷ lệ cache hit trong mô hình bộ nhớ chia sẻ, nâng cao hiệu năng truy xuất bộ nhớ.

2. Áp dụng giải thuật duyệt theo chiều rộng trước đối với tất cả các đỉnh (SSSP) kết hợp với kỹ thuật sử dụng mảng bitmaps để giảm thiểu thời gian truy xuất kiểm tra đỉnh đã duyệt trong hàng đợi.

3. Song song hoá các phép tính SSSP với mô hình lập trình luồng sử dụng bộ thư viện CilkPlus.

4.3.1 Cấu trúc dữ liệu phù hợp

Dữ liệu của đồ thị quy mô lớn $G = (V, E)$ sẽ được biểu diễn theo phương pháp sử dụng các danh sách đỉnh liền kề như đã trình bày ở chương trước: mỗi thành viên sẽ được gán mã định danh trong khoảng từ 0 đến $|V| - 1$. Đối với dữ liệu cạnh, các vectors đỉnh liền kề có sắp xếp sẽ được dùng để lưu thông tin quan hệ giữa các thành viên [DPH2]. Từ đó, dữ liệu cạnh sẽ được biểu diễn trong mảng vectors $Edges[u]$ $\forall u \in V$. Đây là phương pháp tổ chức dữ liệu mang lại khả năng có tỷ lệ cache hit cao nhất khi tham chiếu đến dữ liệu đồ thị.

Các đỉnh đã được duyệt theo phương pháp BFS sẽ được lưu vết trong một vector, mà mỗi bit trong vector sẽ thể hiện trạng thái đã được duyệt hay chưa duyệt của đỉnh tương ứng với vị trí bit ấy.

Đối với hàng đợi Q dùng để lưu vết các đỉnh v sẽ được duyệt, chúng tôi cũng đã tổ chức cấu trúc hàng đợi để ngoài lưu vết đỉnh thì lưu kèm luôn khoảng cách từ đỉnh nguồn s đến đỉnh v tương ứng. Việc tổ chức này cho phép có luôn được giá trị khoảng cách từ s đến v trong bộ nhớ đệm cache mỗi khi xét đỉnh v (tức tăng được tỷ lệ cache hit).

4.3.2 Giải thuật song song tính độ trung tâm gần

Từ giải thuật cơ bản minh hoạ ở 4.1, việc tính độ trung tâm gần CC hoàn toàn dựa vào phương pháp duyệt đồ thị BFS theo chiều rộng trước.

Để giảm kích thước hàng đợi khi duyệt, mỗi khi duyệt BFS đối với đỉnh u, chúng ta sẽ sử dụng mảng $Maps$ mà ở đó vị trí bit thứ v thể hiện trạng thái duyệt hay chưa của đỉnh v. Hàng đợi $Queue$ cũng được tổ chức đặc biệt để lưu luôn được giá trị khoảng cách ngắn nhất từ u đến đỉnh đã duyệt trong $Queue$.

Để có thể khai thác được thế mạnh từ các bộ vi xử lý đa lõi cũng như những hệ thống tính toán đa vi xử lý, chúng tôi sẽ tiến hành tính độ trung tâm gần CC theo phương pháp tính toán song song. Cách tiếp cận tính toán song song của chúng tôi dựa trên việc thi hành song song các phép tính độ trung tâm gần trên các đỉnh khác nhau chứ không phải song song quá trình duyệt và tính đường đi ngắn nhất từ một đỉnh đến tất cả các đỉnh còn lại (SSSP). Cách tiếp cận này cho phép việc duyệt SSSP được thực hiện trong mỗi luồng chuyên biệt, từ đó có thể khai thác được những dữ liệu liền kề nhau trong cache, nâng cao tỷ lệ cache hit.

Do kích thước các hàng đợi $Queue$ lớn (có số phần tử bằng V), thế nên thời gian cấp phát bộ nhớ cho $Queue$ cũng sẽ mất nhiều thời gian. Tương tự như thế đối với mảng đánh dấu duyệt $Maps$. Vì vậy, chúng tôi sẽ cấp phát trước bộ nhớ chứa các mảng này tương ứng với số luồng có thể thi hành song song. Cũng dựa vào đánh giá ở chương trước, bộ thư viện tính toán song song CilkPlus sẽ được sử dụng để cài đặt quá trình tính toán song song độ trung tâm gần.

Quá trình song song tính độ trung tâm gần của chúng tôi được trình bày như trong giải

thuật 4.3 sau:

Thuật toán 4.3: Giải thuật song song tính độ trung tâm gần

Input: $G = (V, E)$ *represented by* $Edges[][]$
Output: $CC[.]$ for all $v \in V$

```
1  CC[v] ← 0, Sum[v] ← 0, Maps[v] ← 0    ∀v ∈ V ;
   /* Thi hành song song sử dụng thư viện Cilk Plus                    */
2  for s = 0 to Edges.size() do
3      Q ← empty queue; Q.push(s);
4      SetBit(s, Maps) ;                          /* Đánh dấu s đã được duyệt */
5      CC[s] ← 0; FC[s] ← 0; dst ← 0 ;
6      while Q is not empty do
7          dst ← dst + 1;
           /* Duyệt các đỉnh cùng mức từ s trong hàng đợi Q              */
8          while Q is not empty do
9              v ← Q.pop();
               /* Duyệt các đỉnh liền kề với s                           */
10             forall w ∈ Edges[s] do
                   /* nếu chưa duyệt                                     */
11                 if !TestBit(w) then
12                     Q.push(w, dst) ;      /* lưu lại khoảng cách từ v đến w */
13                     SetBit(w, Maps) ;              /* đánh dấu w đã duyệt */
14                 end
15             end
16         end
17         Q.nextLevel() ;                          /* chuyển đến mức kế tiếp */
18     end
19     if Sum[s] ≠ 0 then CC[s] ← 1/Sum[s] ;
20 end
21 return CC[.];
```

Có thể thấy, độ phức tạp tính toán về thời gian của giải thuật này $O(|V| * (|V| + |E|))$ tương tự như độ phức tạp của giải thuật cơ bản đã trình bày ở phần trên khi triển khai với một luồng. Trong trường hợp giải thuật này được thi hành song song với t luồng, độ phức tạp thời gian của giải thuật sẽ được giảm đi t lần và có giá trị là $O(\frac{|V|*(|V|+|E|)}{t})$.

4.3.3 Giải thuật song song tính độ trung tâm trung gian

Từ giải thuật cơ bản minh hoạ ở 4.2, việc tính độ trung tâm trung gian BC hoàn toàn dựa vào phương pháp duyệt đồ thị BFS theo chiều rộng trước.

Chúng tôi cũng tiến hành tính độ trung tâm trung gian BC theo phương pháp tính toán

Thuật toán 4.4: Giải thuật song song tính độ trung tâm trung gian

Input: $G = (V, E)$, được tổ chức như mảng vector $Edges[][]$

Data: queue $Q \leftarrow$, stack S khởi tạo rỗng và có thể chứa được $|V|$ đỉnh ;

$dist[v]$: lưu khoảng cách từ đỉnh nguồn đến v ;

$Pred[v]$: chứa danh sách các đỉnh trên đường đi ngắn nhất từ đỉnh nguồn đến v ;

$\sigma[v]$: số đường đi ngắn nhất từ đỉnh nguồn đến v ;

$\delta[v]$: độ thuộc của đỉnh nguồn vào đỉnh v ;

$reducerBC[v]$: vector chứa giá trị BC của đỉnh v cho phép cập nhật tương tranh khi thi hành song song với thư viện Cilk Plus ;

Output: $BC[.]$ với mọi $v \in V$

`/* Thi hành song song sử dụng thư viện Cilk Plus */`

1 **for** $s = 0$ **to** $Edges.size()$ **do**

`/* Pha 1. Duyệt theo kiểu bài toán SSSP */`

2 **foreach** $v \in V$ **do** $Pred[v] \leftarrow$ empty list ;

3 **foreach** $v \in V$ **do** $dist[v] \leftarrow \infty; \sigma[v] \leftarrow 0$;

4 $dist[s] \leftarrow 0; \sigma[s] \leftarrow 1; Q.push(s)$;

5 **while** Q *not empty* **do**

6 $v \leftarrow Q.pop(); S.push(v)$;

7 **foreach** $w \in Edges[v]$ **do**

`/* w chưa được duyệt */`

8 **if** $dist[w] == \infty$ **then** $dist[w] \leftarrow dist[v] + 1; Q.push(w)$;

`/* (v,w) nằm trên đường đi ngắn nhất */`

9 **if** $dist[w] == dist[v] + 1$ **then** $\sigma[w] \leftarrow \sigma[w] + \sigma[v]; Pred[w].push\_back(v)$;

10 **end**

11 **end**

`/* Pha 2: Tích luỹ (tính ngược lại đỉnh đã được duyệt) */`

12 **foreach** $v \in V$ **do** $\delta[v] \leftarrow 0$;

13 **while** S *not empty* **do**

14 $w \leftarrow S.pop()$;

15 **for** $v \in Pred[w]$ **do** $\delta[v] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]}.(1 + \delta[w])$;

`/* Đảm bảo thi hành tương tranh chính xác */`

16 **if** $w \neq s$ **then** $reducerBC[w] \leftarrow reducerBC[w] + \delta[w]$;

17 **end**

18 **end**

19 $reducerBC.move\_out(BC)$; `/* Trả kết quả về cho vector` BC `*/`

20 **return** $BC[.]$;

song song để có thể khai thác được thế mạnh từ các bộ vi xử lý đa lõi cũng như những hệ thống tính toán đa vi xử lý, cụ thể là thi hành song song các phép tính độ trung tâm trung gian trên các đỉnh khác nhau.

Việc cấp phát trước bộ nhớ chứa mảng Q cũng được thực hiện tương tự như đối với giải thuật tính độ trung tâm gần. Bộ thư viện tính toán song song CilkPlus cũng được sử dụng để cài đặt quá trình tính toán song song độ trung tâm trung gian.

Quá trình tính độ trung tâm trung gian sẽ được thi hành song song như minh họa trong giải thuật 4.4.

Tương tự như khi tính độ phức tạp với giải thuật song song tính độ trung tâm gần, giải thuật này này khi thi hành song song với t luồng thì độ phức tạp tính toán là $O(\frac{|V|*|E|}{t})$.

4.4 Thực nghiệm và đánh giá

Để có thể đánh giá cụ thể hơn về giải thuật song song tính hai độ trung tâm gần CC và độ trung tâm trung gian BC, chúng tôi đã tiến hành cài đặt giải thuật, lựa chọn một số bộ công cụ phân tích đồ thị đã bao gồm các hàm tính các độ trung tâm và sử dụng một số bộ dữ liệu mạng xã hội đã được một số tổ chức có uy tín công bố.

4.4.1 Môi trường thử nghiệm, đánh giá

Giải thuật song song tính độ trung tâm gần CC của chúng tôi đã được cài đặt bằng ngôn ngữ C++ trên môi trường tính toán của Trường Đại học Công nghệ, với cấu hình 2 x Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz (45MB Cache, 18-cores per CPU), bộ nhớ chính 128GB, CentOS Linux release 7.2.1511, gcc 6.3.0. Hệ thống tính toán này được cấu hình cho phép thi hành tối đa 36 luồng song song (do cấu hình tắt chức năng hyperthreading trên 2 CPU).

4.4.2 Dữ liệu thực nghiệm

Để kiểm nghiệm giải thuật trên, chúng tôi đã thu thập các bộ dữ liệu mạng xã hội được công bố từ hai tổ chức lớn: SNAP (Stanford Large Network Dataset Collection) [61] và Aminer Datasets for Social Network Analysis [113]. Các bộ dữ liệu được sử dụng trong thực nghiệm đánh giá của chúng tôi gồm:

- *gemsec-Facebook*: Dây là bộ dữ liệu gồm tám mạng con được xây dựng để biểu diễn những trang Facebook đã được kiểm tra chính thống (blue verified Facebook pages). Các trang Facebook này được mô hình hoá bằng các đỉnh đồ thị trong khi các cạnh thể hiện liên kết giữa các trang đó. Do giới hạn thời gian nên chúng tôi chỉ chọn hai mạng lớn trong bộ dữ liệu *gemsec-Facebook* để tiến hành thử nghiệm là: Politician and Artist.

- *ego-Facebook*: Đây là bộ dữ liệu được xây dựng từ các danh sách bạn (friends lists) của Facebook. Các danh sách này được thu thập từ các thành viên tham gia khảo sát dựa trên ứng dụng Facebook.
- *com-DBLP*: Đây là bộ dữ liệu thể hiện mạng đồng tác giả DBLP.
- *com-Youtube*: Đây là bộ dữ liệu thu thập từ cộng đồng trong mạng xã hội Youtube.
- *Flickr*: Đây là bộ dữ liệu thu thập từ mạng chia sẻ ảnh Flickr thể hiện quan hệ giữa các thành viên cộng đồng này.

Trong số các bộ dữ liệu trên, *Flickr* là bộ dữ liệu thể hiện đồ thị không kết nối trong khi các bộ dữ liệu còn lại đều thể hiện là đồ thị có kết nối. Thông tin miêu tả cụ thể các tham số chính thể hiện các bộ dữ liệu này minh hoạ ở bảng 4.1 dưới đây:

Bảng 4.1: Thông tin thống kê về các dữ liệu mạng xã hội thử nghiệm

Bộ dữ liệu	**Số cạnh**	**Số đỉnh**	**Đường kính**
ego-Facebook (DS1)	88.234	4.039	8
gemsec-Facebook Politician (gọi tắt DS2)	41.729	5.908	14
gemsec-Facebook Artist (DS3)	819.306	50.515	11
DBLP (DS4)	1.049.866	425.957	23
Youtube (DS5)	2.987.624	1.157.828	24
Flickr (DS6)	9.114.557	214.626	10

4.4.3 Kết quả thực nghiệm và đánh giá

4.4.3.1 Giải pháp nâng cao hiệu năng tính độ trung tâm gần

Từ mã nguồn của giải pháp bigGraph đã được chúng tôi đề cập ở chương 2 [1], chúng tôi đã tiến hành cài đặt giải pháp song song hoá quá trình tính độ trung tâm gần trên đồ thị mạng xã hội bằng ngôn ngữ C++ và sử dụng thư viện lập trình song song theo mô hình luồng CilkPlus. Toàn bộ mã nguồn của giải pháp này cũng như các kết quả thực nghiệm mà chúng tôi đã tiến hành đều được công bố tại địa chỉ GitHub: `https://github.com/hanhdp/parallel_closeness_centrality/`.

Để đánh giá hiệu năng của giải pháp mà chúng tôi đã xây dựng trong bigGraph, hai bộ công cụ tiêu biểu đã được chúng tôi giới thiệu ở phần Mở đầu được chúng tôi chọn để so sánh là: TeexGraph và NetworKit. Cả hai bộ công cụ này và bigGraph đều được cài đặt trên hạ tầng phần cứng đã được đề cập ở phần trên.

Để phân tích hệ số tăng tốc (speedup), chúng tôi đã tiến hành đánh giá trước tiên giải pháp bigGraph với số lượng luồng thi hành song song thay đổi từ 1 (tương ứng với trường

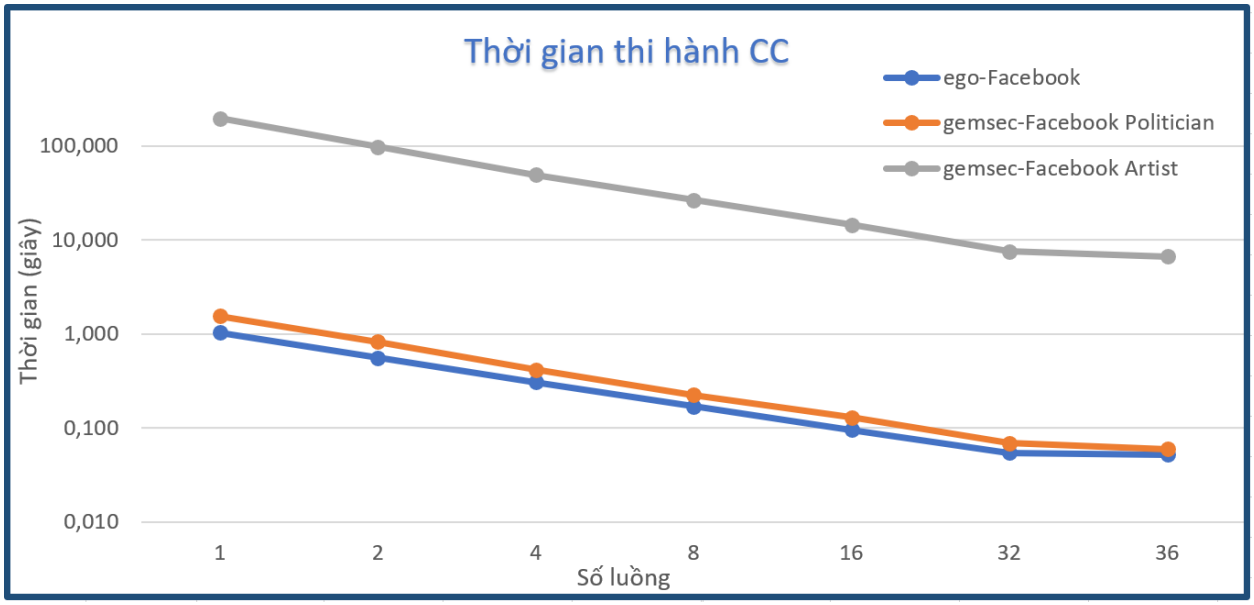
[1] `https://github.com/hanhdp/bigGraph/`

hợp thi hành tuần tự) đến số luồng tối đa (36 luồng) có thể thi hành song song trong hệ thống tính toán. Với mỗi bộ dữ liệu, chúng tôi tiến hành tính độ trung tâm gần 10 lần cho tất cả các đỉnh. Đối với những bộ dữ liệu đồ thị quy mô lớn như Youtube, DBLP và Flickr, thời gian thi hành để tính được độ trung tâm gần nhìn chung là rất cao như minh hoạ ở bảng 4.3. Từ đó, việc so sánh giữa bigGraph và hai bộ công cụ phân tích mạng xã hội còn lại (TeexGraph và NetworKit) sẽ tập trung sử dụng ba bộ dữ liệu đầu: *ego-Facebook-DS1*; *gemsec-Facebook Politician-DS2*; và *gemsec-Facebook Artist-DS3*. Kết quả thực nghiệm của chúng tôi được tổng hợp dựa trên việc tính thời gian thi hành trung bình của cả 10 lần chạy thử nghiệm đối với mỗi giải pháp và được thể hiện ở bảng sau:

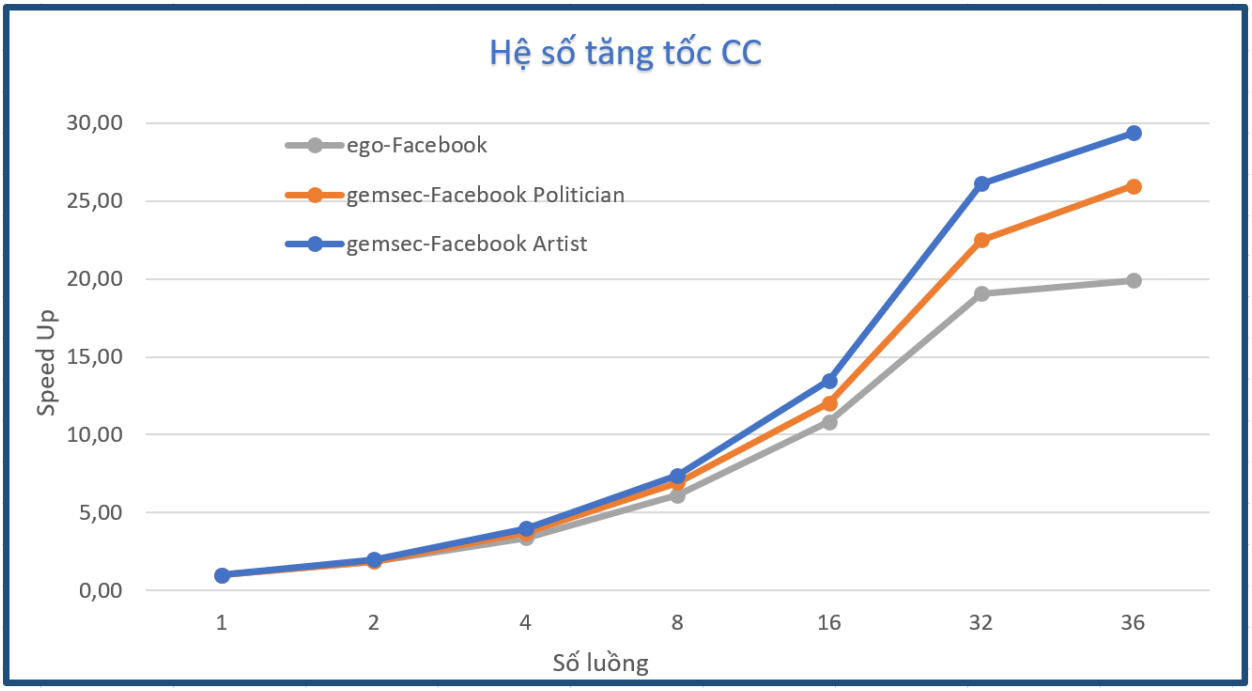
Bảng 4.2: Thời gian (giây) và hệ số tăng tốc của bigGraph khi tính độ trung tâm gần

Số luồng	**DS1**		**DS2**		**DS3**	
	Thời gian	Speedup	Thời gian	Speedup	Thời gian	Speedup
1	1,031	1,00	1,546	1,00	195,276	1,00
2	0,552	1,87	0,819	1,89	97,527	2,00
4	0,306	3,37	0,415	3,72	49,136	3,97
8	0,169	6,11	0,223	6,95	26,418	7,39
16	0,095	10,82	0,129	12,03	14,491	13,48
32	0,054	19,07	0,069	22,48	7,481	26,10
36	0,052	19,89	0,060	25,96	6,648	29,37

Hình sau thể hiện rõ hơn sự thay đổi của hệ số tăng tốc speedup của bigGraph khi số luồng song song thay đổi:



Hình 4.1: Thời gian thi hành bigGraph khi tính độ trung tâm trung gian



Hình 4.2: Đánh giá hệ số tăng tốc bigGraph khi tính độ trung tâm trung gian

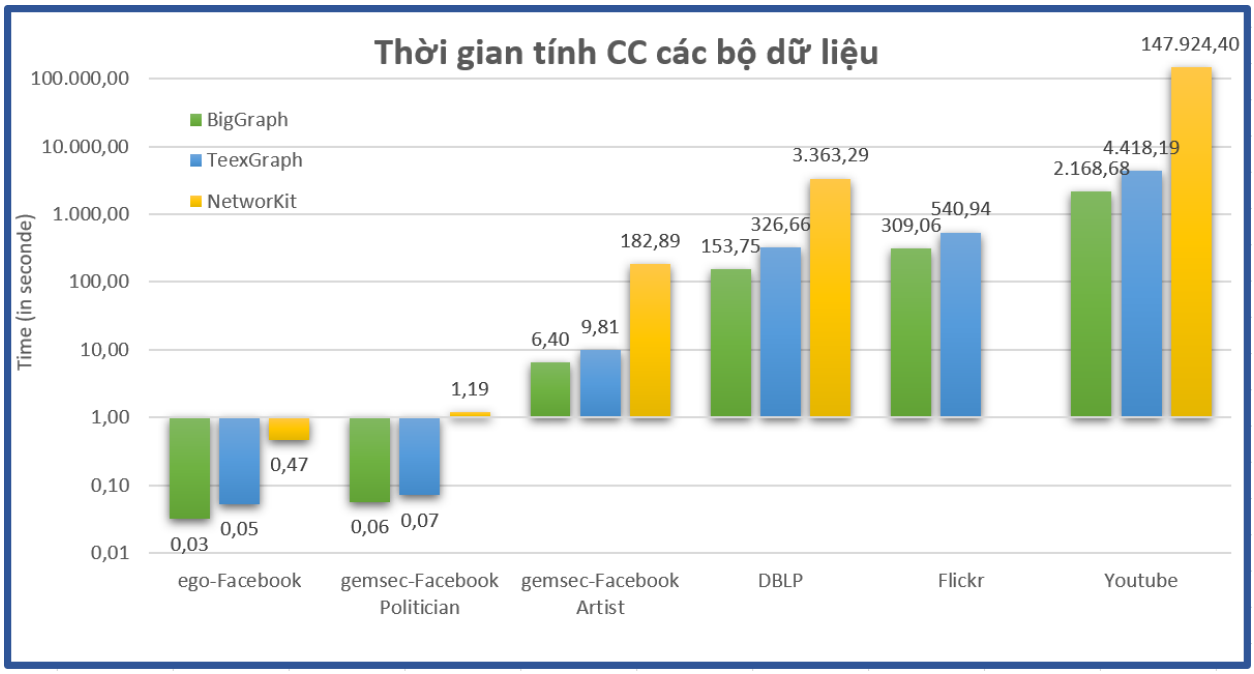
Như minh hoạ ở hình 4.2, càng nhiều luồng song song chúng ta có thể tiến hành, thời gian thi hành tính độ trung tâm gần càng ngắn. Từ đó, chúng tôi quyết định chọn số luồng song song là 36 luồng (số luồng tối đa trên hệ thống tính toán của chúng tôi) khi thi hành so sánh đánh giá hiệu năng giữa bigGraph và NetworKit, TeexGraph.

Bảng sau minh hoạ thời gian trung bình của 10 lần thi hành quá trình tính độ trung tâm gần của cả ba giải pháp nêu trên:

Bảng 4.3: Thời gian tính độ trung tâm gần (giây)

Bộ dữ liệu	**NetworKit**	**TeexGraph**	**bigGraph**
ego-Facebook	0,468	0,052	**0,032**
gemsec-Facebook Politician	1,192	0,071	**0,056**
gemsec-Facebook Artist	182,890	9,808	**6,405**
DBLP	3363,286	326,659	**153,753**
Flickr		540,944	**309,058**
Youtube	147.924,400	4.418,191	**2.168,677**

Trong bảng này, ngoại trừ bộ dữ liệu Flickr là đồ thị không kết nối nên NetworKit không thể tính độ trung tâm gần. Các công cụ khác đều thi hành và trả về độ trung tâm gần một cách chính xác với tất cả các bộ dữ liệu.



Hình 4.3: Thời gian thi hành thực nghiệm tính độ trung tâm gần

Kết quả thực nghiệm thu được cho phép khẳng định giải pháp song song để tính độ trung tâm gần của chúng tôi trong bigGraph có hiệu năng tốt hơn so với những bộ công cụ còn lại. Bảng 4.4 cho thấy hệ số tăng tốc của bigGraph đối với cả 6 bộ dữ liệu đều nhanh hơn từ 1,27 đến 2,12 và từ 14,78 đến 68,21 so với 2 bộ công cụ còn lại, lần lượt là TeexGraph và NetworKit.

Bảng 4.4: Hệ số tăng tố của bigGraph so với TeexGraph và NetworKit khi tính độ trung tâm gần

Bộ dữ liệu	**TeexGraph/bigGraph**	**NetworKit/bigGraph**
ego-Facebook	1,66	14,78
gemsec-Facebook Politician	1,27	21,23
gemsec-Facebook Artist	1,53	28,56
DBLP	2,12	21,87
Flickr	1,75	
Youtube	2,04	68,21

Đối với việc đánh giá tính đúng đắn của ba bộ công cụ này khi thi hành song song, chúng tôi đã sử dụng bộ công cụ NetworkX để tính tuần tự độ trung tâm gần của các đồ thị sử dụng trong thực nghiệm và lưu ra tệp. Tệp này được sử dụng để so khớp với kết quả tính độ trung tâm gần của cả bigGrap, TeexGraph và NetworKit thi hành song song với 36 luồng. Kết quả thực nghiệm khẳng định cả ba giải pháp này đều có kết quả trùng khớp với kết quả đã sinh ra từ công cụ NetworkX.

Như vậy, thực nghiệm với tất cả sáu bộ dữ liệu, giải pháp bigGraph đều thi hành quá trình tính độ trung tâm gần trong thời gian ngắn hơn. Kết quả này minh chứng hiệu năng của bigGraph đã được cải thiện so với những bộ công cụ phân tích mạng xã hội điển hình hiện nay. Hiệu năng bigGraph được cải thiện rõ ràng dựa vào (i) cấu trúc dữ liệu đồ thị phù hợp (cho phép giảm thời gian truy cập dữ liệu đồ thị trong bộ nhớ chính nhờ tổ chức liền kề và sử dụng mảng bitmaps để đánh dấu các đỉnh đã duyệt) và (ii) phương thức song song hoá giải thuật BFS dựa vào bộ thư viện CilkPlus như chúng tôi đã phân tích trong mục 4.3.2.

Các kết quả này của chúng tôi đã được công bố trong kỷ yếu hội thảo SoICT năm 2018 (cũng nằm trong chỉ mục SCOPUS và WoS) [DPH4].

4.4.3.2 Giải pháp nâng cao hiệu năng tính độ trung tâm trung gian

Từ mã nguồn của giải pháp bigGraph với giải pháp song song giải thuật tính độ trung tâm gần nêu trên, chúng tôi đã tiếp tục cài đặt giải pháp song song hoá quá trình tính độ trung tâm trung gian bằng ngôn ngữ C++ và sử dụng thư viện lập trình song song theo mô hình luồng CilkPlus. Toàn bộ mã nguồn của giải pháp này cũng như các kết quả thực nghiệm mà chúng tôi đã tiến hành đều được công bố tại địa chỉ GitHub: `https://github.com/hanhdp/parallel_betweenness_centrality/`.

Việc đánh giá hiệu năng của giải pháp mà chúng tôi đã xây dựng trong bigGraph sẽ được so sánh với cả hai bộ công cụ tiêu biểu trong phân tích đồ thị mạng xã hội quy mô lớn là TeexGraph và NetworKit. Cả hai bộ công cụ này và bigGraph đều được cài đặt trên hạ tầng phần cứng đã được đề cập ở phần trên.

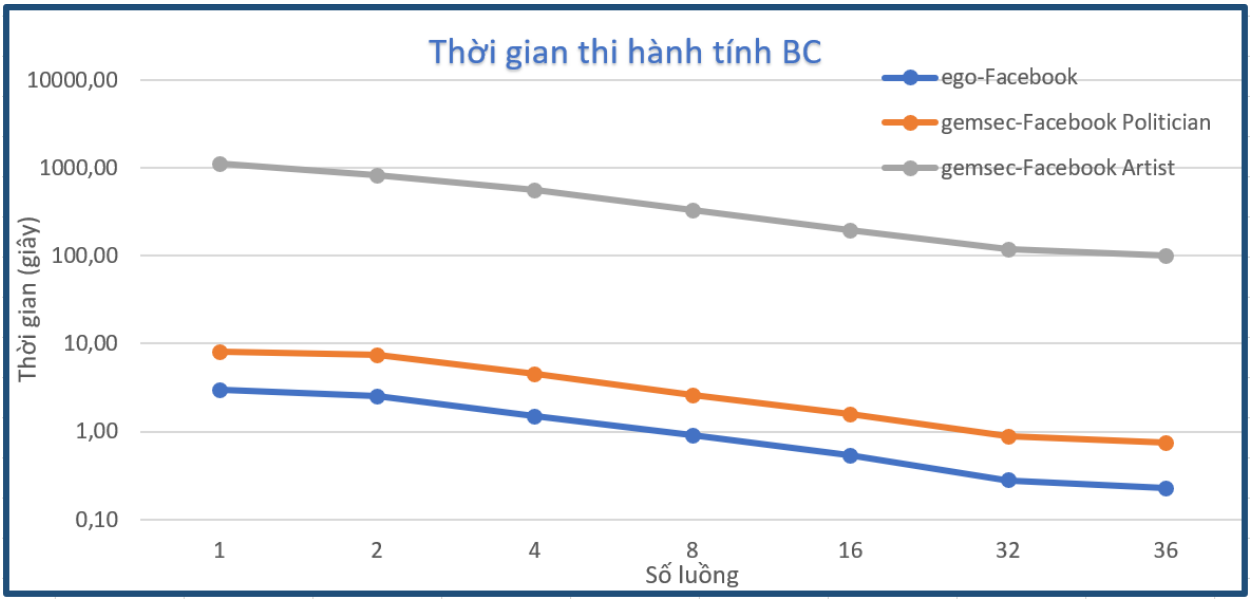
Cũng tương tự như việc đánh giá tính đúng đắn với giải thuật tính độ trung tâm gần, chúng tôi cũng đã sử dụng bộ công cụ NetworkX để tính tuần tự độ trung tâm trung gian của cả 6 đồ thị sử dụng trong thực nghiệm và lưu ra tệp. Dựa trên kết quả thu được khi tính độ trung tâm trung gian của cả bigGrap, TeexGraph và NetworKit thi hành song song với 36 luồng đối với 6 bộ dữ liệu. Kết quả thực nghiệm khẳng định cả ba giải pháp này đều có kết quả tính độ trung tâm trung gian trùng khớp với kết quả đã sinh ra từ công cụ NetworkX.

Để phân tích hệ số tăng tốc (speedup), chúng tôi đã tiến hành đánh giá trước tiên giải pháp bigGraph với số lượng luồng thi hành song song thay đổi từ 1 (tương ứng với trường hợp thi hành tuần tự) đến số luồng tối đa (36 luồng) có thể thi hành song song trong hệ thống tính toán. Đối với những bộ dữ liệu đồ thị quy mô lớn như Youtube, DBLP và Flickr, thời gian thi hành để tính được độ trung tâm trung gian nhìn chung rất cao như minh hoạ ở bảng 4.6. Từ đó, việc so sánh giữa bigGraph và hai bộ công cụ phân tích mạng xã hội còn lại (TeexGraph và NetworKit) sẽ tập trung sử dụng ba bộ dữ liệu đầu: DS1, DS2 và DS3. Kết quả thực nghiệm của chúng tôi được tổng hợp dựa trên việc tính thời gian thi hành trung bình của cả 10 lần chạy thử nghiệm đối với mỗi giải pháp và được thể hiện ở bảng sau:

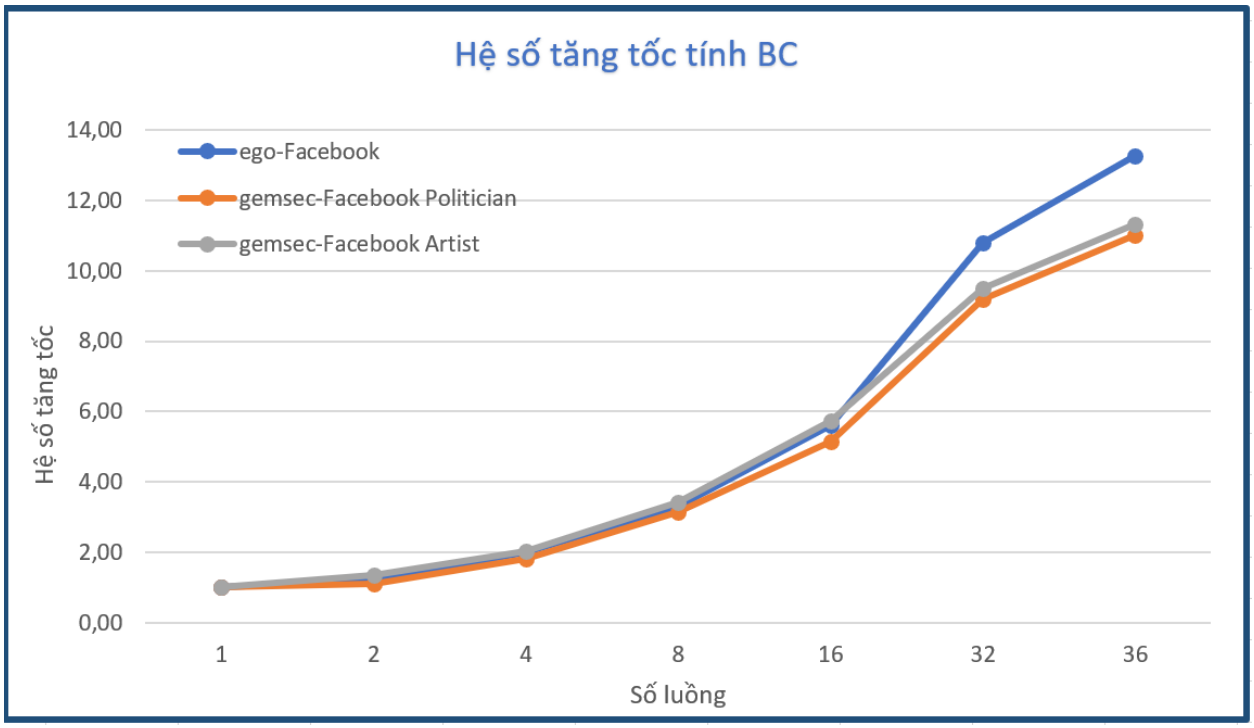
Bảng 4.5: Thời gian (giây) và hệ số tăng tốc của bigGraph khi tính độ trung tâm trung gian

Số luồng	**DS1**		**DS2**		**DS3**	
	Thời gian	Speedup	Thời gian	Speedup	Thời gian	Speedup
1	3,03	1,00	8,20	1,00	1129,47	1,00
2	2,52	1,20	7,44	1,10	832,76	1,36
4	1,51	2,01	4,52	1,82	556,46	2,03
8	0,92	3,29	2,61	3,15	330,64	3,42
16	0,54	5,62	1,60	5,14	196,46	5,75
32	0,28	10,79	0,89	9,19	118,92	9,50
36	0,23	13,26	0,74	11,01	99,85	11,31

Hình sau thể hiện rõ hơn sự thay đổi của hệ số tăng tốc speedup của bigGraph khi số luồng song song thay đổi:



Hình 4.4: Thời gian thi hành tính độ đo BC của bigGraph (giây)



Hình 4.5: Đánh giá hệ số tăng tốc tính độ đo BC của bigGraph

Như minh hoạ ở hình 4.5, càng nhiều luồng song song chúng ta có thể tiến hành, thời gian thi hành tính độ trung tâm trung gian càng ngắn. Từ đó, chúng tôi quyết định chọn số luồng song song là 36 luồng (số luồng tối đa trên hệ thống tính toán của chúng tôi) khi thi hành so sánh đánh giá hiệu năng giữa bigGraph và NetworKit, TeexGraph.

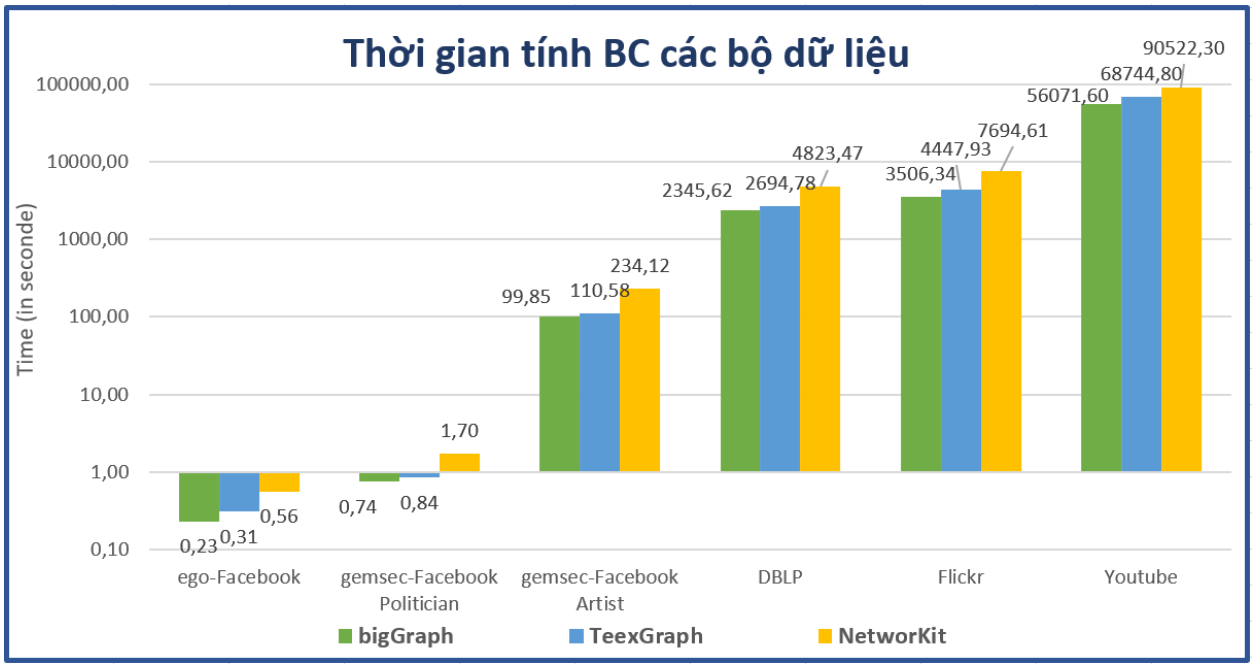
Bảng sau minh hoạ thời gian trung bình của 10 lần thi hành quá trình tính độ trung tâm trung gian của cả ba giải pháp nêu trên:

Bảng 4.6: Thời gian tính độ đo BC (giây)

Bộ dữ liệu	**bigGraph**	**TeexGraph**	**NetworKit**
ego-Facebook	**0,23**	0,31	0,56
gemsec-Facebook Politician	**0,84**	0,84	1,70
gemsec-Facebook Artist	**99,85**	110,58	234,12
DBLP	**2345,62**	2694,78	4823,47
Flickr	**3.506,34**	4.447,93	7.694,61
Youtube	**56.071,60**	68.744,80	90.522,30

Ngoài việc so sánh, đánh giá về thời gian, chúng tôi cũng đã tiến hành so sánh đánh giá kết quả độ trung tâm trung gian thu được của cả ba giải pháp. Kết quả là cả ba giải pháp đều trả về danh sách độ trung tâm trung gian giống nhau với cả 6 bộ dữ liệu nêu trên. Bảng kết quả thực nghiệm 4.6 này minh chứng rõ nét giải thuật tính độ trung tâm trung gian mà chúng tôi xây dựng trong luận án có thời gian thi hành nhỏ hơn so với hai bộ công cụ

TeexGraph và NetworKit. Minh hoạ chi tiết về thời gian thi hành của cả ba công cụ này được thể hiện ở hình 4.6 dưới đây:



Hình 4.6: Thời gian thi hành thực nghiệm tính độ đo BC

Kết quả thực nghiệm thu được cho phép khẳng định được giải pháp song song tính độ trung tâm trung gian của chúng tôi trong bigGraph có hiệu năng tốt hơn so với những bộ công cụ còn lại. Bảng 4.4 cho thấy hệ số tăng tốc của bigGraph đối với cả 6 bộ dữ liệu đều nhanh hơn từ 1,11 đến 1,35 và từ 2,06 đến 2,44 so với hai công cụ còn lại, lần lượt là TeexGraph và NetworKit.

Bảng 4.7: Hệ số tăng tốc của bigGraph so với TeexGraph và NetworKit khi tính độ đo BC

Bộ dữ liệu	**TeexGraph/bigGraph**	**NetworKit/bigGraph**
ego-Facebook	1,35	2,44
gemsec-Facebook Politician	1,13	2,28
gemsec-Facebook Artist	1,11	2,34
DBLP	1,15	2,06
Flickr	1,27	1,61
Youtube	1,23	2,19

Từ kết quả thực nghiệm trên, có thể thấy với tất cả sáu bộ dữ liệu, giải pháp bigGraph đều thi hành quá trình tính độ trung tâm trung gian trong thời gian ngắn hơn. Kết quả này minh chứng hiệu năng của bigGraph đã được cải thiện so với những bộ công cụ phân tích đồ thị điển hình hiện nay. Cũng tương tự như với giải pháp nâng cao hiệu năng tính độ trung tâm gần, giải pháp tính độ trung tâm trung gian của chúng tôi có hiệu năng tốt hơn

so với NetworKit và TeexGraph đều dựa vào (i) cấu trúc dữ liệu đồ thị phù hợp để giảm thời gian truy cập dữ liệu đồ thị trong bộ nhớ chính và (ii) phương thức song song hoá giải thuật Brandes dựa vào bộ thư viện CilkPlus. Kết quả này đang được chúng tôi gửi xét đăng trên tạp chí chuyên ngành trong thời gian tới.

4.5 Kết chương 4

Trong chương này, chúng tôi đã tập trung nghiên cứu nâng cao hiệu năng quá trình tính toán một số độ đo trung tâm trên đồ thị. Hai độ đo trung tâm quan trọng trong phân tích đồ thị đã được chúng tôi lựa chọn nghiên cứu sâu trong luận án này là: (i) độ trung tâm gần và (ii) độ trung tâm trung gian.

Từ các khảo sát, đánh giá các nghiên cứu liên quan đến việc tính hai độ đo này, chúng tôi đã tiến hành nghiên cứu, đề xuất hai giải pháp để tính hai độ trung tâm đó trên đồ thị không trọng số (cả có hướng và vô hướng). Cả hai giải pháp đều dựa trên những giải thuật tính độ trung tâm có độ phức tạp tính toán tốt nhất hiện nay, từ đó thực hiện kết hợp (i) tổ chức dữ liệu đồ thị phù hợp và (ii) kỹ thuật tính toán song song. Đối với quá trình tính độ trung tâm gần, giải pháp chúng tôi xây dựng có độ phức tạp là $O(\frac{|V|*(|V|+|E|)}{t})$. với t là số luồng có thể thi hành song song. Đối với giải pháp tính độ trung tâm trung gian, độ phức tạp tính toán của giải thuật được cài đặt là $O(\frac{|V|*|E|}{t})$. Các thực nghiệm chúng tôi tiến hành với các bộ dữ liệu đồ thị được cung cấp từ các tổ chức nổi tiếng (gồm 6 bộ dữ liệu) đã cho phép khẳng định được hiệu quả vượt trội của hai giải pháp chúng tôi xây dựng trong luận án so với một số bộ công cụ phân tích đồ thị như NetworKit và TeexGraph.

Chiến lược tính hai độ trung tâm đã trình bày ở trên hoàn toàn có thể mở rộng để nâng cao hiệu năng tính các độ trung tâm còn lại như độ trung tâm vector riêng, độ trung tâm pagerank, ...

Chương 5

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1 Các đóng góp chính

Thực tế đã chứng minh được tính hiệu quả của lý thuyết đồ thị trong những bài toán thực tiễn, chẳng hạn như mô hình hoá các phép toán trên mạng xã hội Facebook, Twitter... Luận án này quan tâm đến việc ứng dụng lý thuyết đó để giải quyết bài toán quan trọng: nâng cao hiệu năng xử lý các phép toán đồng thời trên đồ thị. Với định hướng đó, chúng tôi đã tiến hành xác lập mục tiêu và các nội dung nghiên cứu chính của luận án (đã trình bày ở Chương 1). Qua các kết quả cả về lý thuyết lẫn thực nghiệm đã được trình bày cụ thể trong luận án, có thể khẳng định rằng toàn bộ mục tiêu và các nội dung nghiên cứu đề ra đã được hoàn thành, với các đóng góp chính bao gồm:

1. Mô hình hoá quá trình xử lý các phép toán tương tranh trên đồ thị quy mô lớn dựa vào lịch thi hành các phép toán đồng thời và dựa vào cấu trúc dữ liệu phù hợp; từ đó cho phép nâng cao hiệu năng bộ nhớ đệm cache và giảm thời gian truy xuất đến bộ nhớ chính.

2. Đề xuất ba giải pháp (akGroup, akGroupPlus và bigGraph) để nâng cao hiệu năng thi hành các truy vấn đồng thời trên đồ thị động quy mô lớn với khả năng thi hành song song cả các truy vấn duyệt đồ thị lẫn cập nhật đồ thị.

3. Nâng cao hiệu năng quá trình tính hai độ đo trung tâm: độ trung tâm gần và độ trung tâm trung gian trên đồ thị quy mô lớn, với giải pháp bigGraph được xây dựng dựa trên việc tổ chức dữ liệu đồ thị phù hợp và song song hoá các phép tính SSSP trên mỗi thành viên của mạng.

Trong luận án, chúng tôi đã đặc tả bài toán xử lý các truy vấn khoảng cách ngắn nhất trên đồ thị động, cách tiếp cận chia các phép toán trên đồ thị thành hai lớp: (i) các phép

toán cập nhật bao gồm thêm, xoá cạnh (ngay cả đối với đồ thị thuộc tính, việc thay đổi thuộc tính nào đó trong đồ thị cũng có thể quy về phép toán xoá cạnh rồi bổ sung thêm cạnh có thuộc tính mới chẳng hạn); và (ii) các phép toán truy vấn trên đồ thị, được tiến hành thông qua các giải thuật duyệt đồ thị hoặc xác định đường đi ngắn nhất giữa các đỉnh. Với việc tập trung nghiên cứu các đồ thị không trọng số, có hướng và có quy mô đỉnh/cạnh lớn, bài toán xử lý các phép toán tương tranh được chúng tôi chú trọng đến trường hợp các phép toán cập nhật và truy vấn khoảng cách ngắn nhất giữa các đỉnh được gửi đến liên tục, đồng thời trên đồ thị. Lúc này, các phép toán đó cần phải được thi hành với điều kiện đảm bảo được tính toàn vẹn, nhất quán của dữ liệu đồ thị. Từ đó, các phép toán tương tranh trong đồ thị được mô hình hoá như một lịch thi hành các phép toán S trên đồ thị G.

Để giải quyết bài toán đó, việc tiến hành thực thi lịch S được chúng tôi đề xuất thông qua ba giải pháp khác nhau. Cả ba giải pháp này đều dựa trên ý tưởng chính là (i) xây dựng cấu trúc dữ liệu đồ thị phù hợp để nâng cao hiệu năng của bộ nhớ đệm cache; (ii) lựa chọn hướng duyệt đồ thị một cách linh hoạt dựa không chỉ vào số lượng đỉnh con mà cả số lượng đỉnh cháu của mỗi hàng đợi; và (iii) đề xuất giải pháp song song hoá các truy vấn đồng thời, cả đối với các phép toán cập nhật lẫn truy vấn khoảng cách ngắn nhất trên đồ thị. Với các giải pháp được đề xuất trong luận án, các giải thuật song song đều đảm bảo tính cục bộ dữ liệu khi thi hành trong mỗi luồng, từ đó tránh sự trao đổi giữa chúng và không cần sử dụng cơ chế kiểm soát tương tranh.

Trong giải pháp 1 (akGroup), dữ liệu đồ thị được tổ chức trên hai mảng chứa toàn bộ các đỉnh liền kề và hai mảng chỉ mục để lưu số đỉnh liền kề và vị trí bắt đầu trong mảng dữ liệu đồ thị. Cách tổ chức này được tiến hành đối với cả chiều đi (outgoing) và chiều đến (incoming) để cho phép có thể tiến hành tính khoảng cách ngắn nhất dựa theo duyệt BFS cả hai chiều. Mặc dù cách thức tổ chức dữ liệu có tính cục bộ cao, nâng cao được tỷ lệ cache hit, nhưng các phép toán cập nhật (đặc biệt thao tác thêm cạnh) lại có hiệu năng không cao. Kết quả này được chúng tôi công bố trong công trình [DPH1] tại hội thảo BDCAT về quản lý dữ liệu lớn năm 2016.

Từ các điểm hạn chế ở giải pháp 1, chúng tôi đã đề xuất giải pháp 2 (akGroupPlus) với mô hình tổ chức dữ liệu đồ thị khác đi. Với phương pháp nhúng kèm trạng thái cạnh đồ thị (sử dụng 2 bits cuối của định danh đỉnh liền kề), các phép toán cập nhật được chia làm hai giai đoạn: chuyển trạng thái các cạnh có thao tác cập nhật (thêm/xoá) thành UNKNOWN và sau khi hoàn thành các truy vấn tính khoảng cách xong mới ghi nhận thực sự các cạnh đó với các trạng thái thêm (ALIVE) hay bị xoá đi (DEAD). Với cách tổ chức dữ liệu này, việc thi hành toàn bộ các truy vấn khoảng cách trong S được tập hợp lại và xử lý song song trên hệ thống tính toán. Từ đó, hiệu năng của quá trình xử lý S đã được cải thiện như đã được minh chứng trong thực nghiệm thứ 2 của chúng tôi. Mặc dù vậy, mới chỉ có các truy vấn khoảng cách trong S được tiến hành song song trong khi các phép toán cập nhật vẫn chỉ được xử lý tuần tự. Giải pháp này của chúng tôi cũng đã được công bố tại hội thảo quốc tế ICCCI năm 2017 [DPH2].

Trong giải pháp 3 (bigGraph, chúng tôi đã đề xuất kỹ thuật cho phép tiến hành song song các phép toán cập nhật với ý tưởng có thể tiến hành song song các phép toán cập nhật trên các danh sách đỉnh liền kề khác nhau. Từ ý tưởng đó, chúng tôi đã xây dựng phương án tiến hành song song các phép toán cập nhật. Các thực nghiệm được chúng tôi tiến hành với giải pháp này cũng đã minh chứng được hiệu quả của quá trình song song cả các phép toán cập nhật lẫn tính khoảng cách ngắn nhất. Kết quả này cũng được chúng tôi công bố trong tạp chí quốc tế Transactions on Computational Collective Intelligence, Springer, năm 2018 [DPH3].

Đối với các phép toán phân tích đồ thị, trong luận án này, chúng tôi đã quan tâm nghiên cứu về các độ đo trung tâm. Với cách tiếp cận tương tự như việc xử lý truy vấn đồng thời nêu trên, chúng tôi đã xây dựng được hai giải thuật tính độ trung tâm gần và độ trung tâm trung gian hiệu quả hơn, với ý tưởng dựa trên việc tổ chức dữ liệu đồ thị hợp lý và song song hoá quá trình xử lý bài toán SSSP. Giải pháp này của chúng tôi, được đặt tên là bigGraph, cũng đã được tiến hành thực nghiệm đánh giá so với một số bộ công cụ khác có cùng miền ứng dụng là TeexGraph và NetworKit. Các kết quả thực nghiệm đánh giá độ trung tâm gần cho thấy giải pháp bigGraph của chúng tôi nhanh hơn TeexGraph và NetworKit lần lượt từ 1,27 đến 2,12 và 14,78 đến 68,21 lần. Kết quả này của chúng tôi đã được công bố trong kỷ yếu hội thảo quốc tế SoICT năm 2018 [DPH4]. Đối với các kết quả đánh giá độ trung tâm trung gian, giải pháp bigGraph cũng cho hiệu năng thi hành tốt hơn so với hai bộ công cụ TeexGraph và NetworKit lần lượt từ 1,11 đến 1,35 và từ 2,06 đến 2,44 lần. Những kết quả này cũng đã được tổng hợp và gửi bài báo xét đăng trong tạp chí chuyên ngành trong thời gian tới.

Toàn bộ các công trình đã công bố của chúng tôi đều được xuất bản trong các kỷ yếu hội thảo, tạp chí có chỉ mục trong SCOPUS và WoS.

5.2 Hạn chế của luận án

Do những hạn chế cả về thời gian, về nguồn lực lẫn sự giới hạn về không gian nghiên cứu, những nghiên cứu trong luận án cũng còn nhiều điểm chưa thể giải quyết hoặc chưa được đề cập đến. Một số điểm hạn chế này có thể được liệt kê dưới đây:

- Các nội dung nghiên cứu của luận án còn chưa thể tiến hành với những đồ thị/mạng xã hội có quy mô siêu lớn, chẳng hạn như Facebook với số lượng đỉnh hơn hai tỷ và hơn nghìn tỷ cạnh: Hạn chế này xuất phát từ giới hạn các bộ dữ liệu thực được công bố công khai quy mô lớn như thế còn hạn chế (các hãng như Facebook cũng không thể cung cấp được toàn bộ dữ liệu đồ thị của họ với cộng đồng). Ngoài ra, năng lực hệ thống tính toán mà nghiên cứu sinh có thể sử dụng cũng không thể cho phép xử lý được với những dữ liệu quy mô siêu lớn như thế.

- Một số độ đo trung tâm như độ trung tâm vector riêng, độ trung tâm xếp hạng trang...

còn chưa được nghiên cứu, cài đặt trong phần tính các độ trung tâm. Đây là hạn chế do thời gian của nghiên cứu sinh chưa có đủ để tiến hành được các nghiên cứu này.

- Các kết quả nghiên cứu của luận án chưa được ứng dụng cụ thể vào các bài toán thực tế: do những giới hạn về nguồn lực nêu trên nên các kết quả của luận án chưa thể tích hợp, hình thành được một số sản phẩm như mong muốn của chúng tôi, chẳng hạn như một hệ quản trị CSDL đồ thị hay một bộ thư viện hoàn chỉnh phục vụ cho các lớp bài toán duyệt và phân tích đồ thị/mạng xã hội.

5.3 Hướng phát triển tương lai

Toàn bộ những hạn chế chế đã nêu trên đều được chúng tôi xác định sẽ là những nghiên cứu được chú trọng tiến hành trong thời gian tới.

Trong thời gian tới, những đồ thị quy mô siêu lớn cũng sẽ được chúng tôi quan tâm nghiên cứu để có thể làm chủ được những công nghệ xử lý đối với quy mô dữ liệu đồ thị siêu lớn đó. Đối với các độ đo trung tâm cũng như các phép toán phân tích đồ thị, chúng tôi cũng sẽ tiến hành cài đặt trong thời gian tới để hình thành nên bộ công cụ hoàn chỉnh phục vụ cho các lớp bài toán duyệt và phân tích đồ thị/mạng xã hội. Ngoài ra, với những tiềm năng của CSDL đồ thị, trong tương lai gần, chúng tôi sẽ hướng đến những nghiên cứu chuyên sâu hơn trong việc tối ưu hoá xử lý các truy vấn trên đồ thị thuộc tính; cho phép xử lý hiệu quả hơn các truy vấn trên đồ thị động, thay đổi theo thời gian (evolving graph database)...

DANH MỤC CÁC CÔNG BỐ CỦA LUẬN ÁN

- [**DPH1**] **Phuong-Hanh DU**, Hai-Dang Pham, Ngoc-Hoa Nguyen, "Optimizing the Shortest Path Query on Large-Scale Dynamic Directed Graph", *BDCAT '16 Proceedings of the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, pp210-216, 2016. (SCOPUS, WoS)

- [**DPH2**] **Phuong-Hanh DU**, Hai-Dang Pham, Ngoc-Hoa Nguyen, "An Efficient Parallel Method for Performing Concurrent Operations on Social Networks", *Computational Collective Intelligence*, Volume 10448 of the series Lecture Notes in Computer Science, Springer, pp 148-159, 2017. (SCOPUS, WoS)

- [**DPH3**] **Phuong-Hanh DU**, Hai-Dang Pham, Ngoc-Hoa Nguyen, "An Efficient Parallel Method for Optimizing Concurrent Operations on Social Networks", *Transactions on Computational Collective Intelligence*. Lecture Notes in Computer Science (Q2), vol 10840, no XXIX, pp. 182-199. Springer 2018, ISSN 2190-9288 (0302-9743). (SCOPUS, WoS)

- [**DPH4**] **Phuong-Hanh DU**, Hai-Chau NGUYEN, Kim-Khoa NGUYEN, and Ngoc-Hoa NGUYEN. "An Efficient Parallel Algorithm for Computing the Closeness Centrality in Social Networks". *In The Ninth International Symposium on Information and Communication Technology (SoICT 2018)*, pp. 456-462, December, 2018. ACM, USA. (SCOPUS, WoS)

Tài liệu tham khảo

[1] S. Adve, V. Adve, G. Agha, M. I Frank María, M. Garzarán, J. Hart, W.-m. Hwu, R. Johnson, K. Rakesh Kumar, D. Marinov, K. Nahrstedt, D. Padua, M. Parthasarathy, S. Patel Grigore Rosu, D. Roth, M. Snir, J. Torrellas, and C. Zilles. Parallel computing research at illinois the upcrc agenda. November 2008.

[2] R. Alhajj and J. Rokne. Encyclopedia of Social Network Analysis and Mining. Springer, 1st edition, 2014. ISBN 978-1-4614-6169-2.

[3] G. S. Almasi and A. Gottlieb. Highly Parallel Computing. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1989. ISBN 0-8053-0177-1.

[4] K. Asanovic, R. Bodik, B. Catanzaro, J. James Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. Plishker, J. Shalf, S. Williams, and K. Yelick. The landscape of parallel computing research: A view from berkeley. EECS Department, University of California, Berkeley, EECS-2006-183:56, 12 2006.

[5] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick. A view of the parallel computing landscape. Commun. ACM, 52(10):56–67, Oct. 2009. ISSN 0001-0782. doi: 10.1145/1562764.1562783. URL `http://doi.acm.org/10.1145/1562764.1562783`.

[6] M. Baglioni, F. Geraci, M. Pellegrini, and E. Lastres. Fast exact computation of betweenness centrality in social networks. In 2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), pages 450–456, 08 2012. ISBN 978-1-4673-2497-7. doi: 10.1109/ASONAM.2012.79.

[7] D. S. Banerjee, S. Sharma, and K. Kothapalli. Work efficient parallel algorithms for large graph exploration. In 20th Annual International Conference on High Performance Computing, pages 433–442, Dec 2013. doi: 10.1109/HiPC.2013.6799125.

[8] B. Barney. Message passing interface (mpi). `https://computing.llnl.gov/tutorials/mpi/`, .

[9] B. Barney. Openmp. `https://computing.llnl.gov/tutorials/openMP/`, .

[10] B. Barney. Posix threads programming. https://computing.llnl.gov/tutorials/pthreads/, .

[11] S. Beame. Understanding and Improving Graph Algorithm Performance. PhD thesis, University of California, Berkeley, 2016.

[12] M. Bentert, A. Dittmann, L. Kellerhals, A. Nichterlein, and R. Niedermeier. An adaptive version of brandes' algorithm for betweenness centrality. Computing Research Repository-CoRR, abs/1802.06701, 2018. URL http://arxiv.org/abs/1802.06701.

[13] M. Bernaschi, G. Carbone, and F. Vella. Scalable betweenness centrality on multi-gpu systems. In Proceedings of the ACM International Conference on Computing Frontiers, CF '16, pages 29–36, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4128-8. doi: 10.1145/2903150.2903153. URL http://doi.acm.org/10.1145/2903150.2903153.

[14] P. Boldi and S. Vigna. Axioms for centrality. Internet Mathematics, 10(3-4):222–262, 2014. doi: 10.1080/15427951.2013.865686.

[15] U. Brandes. A faster algorithm for betweenness centrality. The Journal of Mathematical Sociology, 25(2):163–177, 2001. doi: 10.1080/0022250X.2001.9990249.

[16] E. Bullmore and O. Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. Nature Rev. Neurosci., 10:186–198, 3 2009. doi: 10.1038/nrn2575.

[17] A. Buluç, J. T. Fineman, M. Frigo, J. R. Gilbert, and C. E. Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In Proceedings of the Twenty-first Annual Symposium on Parallelism in Algorithms and Architectures, SPAA '09, pages 233–244, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-606-9. doi: 10.1145/1583991.1584053. URL http://doi.acm.org/10.1145/1583991.1584053.

[18] W. M. Campbell, C. K. Dagli, and C. J. Weinstein. Social network analysis with content and graphs. LINCOLN LABORATORY JOURNAL, 20(1):39–45, 2013.

[19] V. T. Chakaravarthy, F. Checconi, F. Petrini, and Y. Sabharwal. Scalable single source shortest path algorithms for massively parallel systems. In 2014 IEEE 28th International Parallel and Distributed Processing Symposium, pages 889–901, May 2014. doi: 10.1109/IPDPS.2014.96.

[20] M. H. Chehreghani. An efficient algorithm for approximate betweenness centrality computation. The Computer Journal, 57(9):1371–1382, Sept 2014. ISSN 0010-4620. doi: 10.1093/comjnl/bxu003.

[21] D. Chen, L. Lü, M.-S. Shang, Y.-C. Zhang, and T. Zhou. Identifying influential nodes in complex networks. Physica A: Statistical Mechanics and its Applications, 391(4): 1777 – 1787, 2012. ISSN 0378-4371. doi: https://doi.org/10.1016/j.physa.2011.09.017. URL http://www.sciencedirect.com/science/article/pii/S0378437111007333.

[22] Y. Cheng, F. Wang, H. Jiang, Y. Hua, D. Feng, Y. Wu, T. Zhu, and W. Guo. A highly cost-effective task scheduling strategy for very large graph computation. Future Generation Computer Systems, 89:698 – 712, 2018. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2018.07.010. URL http://www.sciencedirect.com/science/article/pii/S0167739X17323683.

[23] B. Chikhaoui, M. Chiazzaro, S. Wang, and M. Sotir. Detecting communities of authority and analyzing their influence in dynamic social networks. ACM Trans. Intell. Syst. Technol., 8(6):82:1–82:28, Aug. 2017. ISSN 2157-6904. doi: 10.1145/3070658. URL http://doi.acm.org/10.1145/3070658.

[24] A. Ching. Scaling apache giraph to a trillion edges. https://www.facebook.com/notes/facebook-engineering/scaling-apache-giraph-to-a-trillion-edges/10151617006153920.

[25] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan. One trillion edges: Graph processing at facebook-scale. Proc. VLDB Endow., 8(12):1804–1815, Aug. 2015. ISSN 2150-8097. doi: 10.14778/2824032.2824077. URL http://dx.doi.org/10.14778/2824032.2824077.

[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms. MIT Lincoln Laboratory Series. The MIT Press, 3rd edition, 2009. ISBN 9780262033848.

[27] S. Das. Efficient algorithms for analyzing large scale network dynamics: Centrality, community and predictability. PhD thesis, Missouri University of Science and Technology, 2018.

[28] E. W. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, 1(1):269–271, 1959. doi: 10.1007/BF01386390.

[29] S. Duggirala. A detailed analysis of nosql and newsql databases for bigdata analytics and distributed computing. In P. Raj and G. C. Deka, editors, A Deep Dive into NoSQL Databases: The Use Cases and Applications, volume 109 of Advances in Computers, chapter 1, pages 1 – 49. Elsevier, 2018. doi: https://doi.org/10.1016/bs.adcom.2018.01.004. URL http://www.sciencedirect.com/science/article/pii/S0065245818300135.

[30] S. Duggirala. Newsql databases and scalable in-memory analytics. In P. Raj and G. C. Deka, editors, A Deep Dive into NoSQL Databases: The Use Cases and Applications, volume 109 of Advances in Computers, chapter 2, pages 49 – 76. Elsevier, 2018. doi: https://doi.org/10.1016/bs.adcom.2018.01.004. URL http://www.sciencedirect.com/science/article/pii/S0065245818300135.

[31] D. Eppstein and J. Wang. Fast approximation of centrality. In Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '01, pages 228–229, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics. ISBN 0-89871-490-7. URL http://dl.acm.org/citation.cfm?id=365411.365449.

[32] D. Eppstein and J. Wang. Fast approximation of centrality. Journal of Graph Algorithms and Applications, 8(1):39–45, 2004. URL http://eudml.org/doc/52454.

[33] S. Even. Graph Algorithms. Cambridge University Press, 2nd edition, 2011. ISBN 0521736536.

[34] R. Fan, K. Xu, and J. Zhao. A gpu-based solution for fast calculation of the betweenness centrality in large weighted networks. PeerJ Computer Science, 3:e140, Dec. 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.140. URL https://doi.org/10.7717/peerj-cs.140.

[35] A. Farooq, G. J. Joyia, M. Uzair, and U. Akram. Detection of influential nodes using social networks analysis based on network metrics. In 2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), pages 1–6, March 2018. doi: 10.1109/ICOMET.2018.8346372.

[36] L. C. Freeman. A set of measures of centrality based on betweenness. Sociometry, 40 (1):35–41, 1977. ISSN 00380431.

[37] L. C. Freeman. Centrality in social networks conceptual clarification. Social Networks, 1(3):215 – 239, 1978. ISSN 0378-8733. doi: https://doi.org/10.1016/0378-8733(78)90021-7. URL http://www.sciencedirect.com/science/article/pii/0378873378900217.

[38] A. V. Goldberg, D. S. J. C. Demetrescu, C. Demetrescu, and D. S. Johnson. The shortest path problem : ninth DIMACS implementation challenge. American Mathematical Society, DIMACS; New ed. edition, 2009. ISBN 0821843834.

[39] M. Gong, G. Li, Z. Wang, L. Ma, and D. Tian. An efficient shortest path approach for social networks based on community structure. CAAI Transactions on Intelligence Technology, 1(1):114 – 123, 2016. ISSN 2468-2322. doi: https://doi.org/10.

1016/j.trit.2016.03.011. URL http://www.sciencedirect.com/science/article/pii/S2468232216000123.

[40] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation, OSDI'12, pages 17–30, Berkeley, CA, USA, 2012. USENIX Association. ISBN 978-1-931971-96-6. URL http://dl.acm.org/citation.cfm?id=2387880.2387883.

[41] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica. Graphx: Graph processing in a distributed dataflow framework. In Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation, OSDI'14, pages 599–613, Berkeley, CA, USA, 2014. USENIX Association. ISBN 978-1-931971-16-4. URL http://dl.acm.org/citation.cfm?id=2685048.2685096.

[42] A. Grama, A. Gupta, G. Karypis, and V. Kuma. Introduction to Parallel Computing. Addison Wesley, 2nd edition, 2003. ISBN 8131708071.

[43] A. Guerrieri. Distributed computing for large-scale graphs: Models, Algorithms, Applications. PhD thesis, Universit'a degli Studi di Trento, 2015.

[44] A. Hagberg, P. Swart, and D. S. Chult. Exploring network structure, dynamics, and function using networkx. In In Proceedings of the 7th Python in Science Conference (SciPy), pages 11–15, 01 2008.

[45] D. Hallac, J. Leskovec, and S. Boyd. Network lasso: Clustering and optimization in large graphs. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '15, pages 387–396, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3664-2. doi: 10.1145/2783258.2783313. URL http://doi.acm.org/10.1145/2783258.2783313.

[46] H_minor_free. Engineering distance queries on highly evolving networks. http://dsg.uwaterloo.ca/sigmod16contest/downloads/H_minor_free-poster.pdf, 2016.

[47] R. C. Holte, A. Felner, G. Sharon, N. R. Sturtevant, and J. Chen. Mm: A bidirectional search algorithm that is guaranteed to meet in the middle. Artificial Intelligence, 252: 232 – 266, 2017. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2017.05.004. URL http://www.sciencedirect.com/science/article/pii/S0004370217300905.

[48] E. E. Ian Robinson, Jim Webber. Graph Databases: New Opportunities for Connected Data. O'Reilly Media, 2nd edition, 2015.

[49] Intel. Cilk plus programming. https://www.cilkplus.org, 2018.

[50] F. Jamour, S. Skiadopoulos, and P. Kalnis. Parallel algorithm for incremental betweenness centrality on large graphs. IEEE Transactions on Parallel and Distributed Systems, 29(3):659–672, March 2018. ISSN 1045-9219. doi: 10.1109/TPDS.2017.2763951.

[51] H. Jeong, S. P. Mason, A.-L. Barabási, and Z. N. Oltvai. Lethality and centrality in protein networks. Nature, 411(6833):41–42, 2001. ISSN 1476-4687. doi: 10.1038/35075138. URL https://doi.org/10.1038/35075138.

[52] U. Kang, S. Papadimitriou, J. Sun, and H. Tong. Centralities in large networks: Algorithms and observations. pages 119–130, 04 2011. doi: 10.1137/1.9781611972818.11.

[53] M. Kaya, J. Kawash, S. Khoury, and M.-Y. Day. Social Network Based Big Data Analysis and Applications. Lecture Notes in Social Networks. Springer International Publishing, 1st edition, 2018. ISBN 978-3-319-78195-2,978-3-319-78196-9.

[54] J. Kim and J.-G. Lee. Community detection in multi-layer graphs: A survey. SIGMOD Rec., 44(3):37–48, Dec. 2015. ISSN 0163-5808. doi: 10.1145/2854006.2854013. URL http://doi.acm.org/10.1145/2854006.2854013.

[55] A. Kouznetsov and M. Tsvetovat. Social Network Analysis for Startups. O'Reilly Media, Inc., 1st edition, 2011. ISBN 9781449311377.

[56] A. Kyrola. Large-scale Graph Computation on Just a PC. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, 2014.

[57] C. Y. Lee. An algorithm for path connections and its applications. IEEE Transactions on Electronic Computers, 10(3):346–65, 2016. doi: 10.1109/TEC.1961.5219222.

[58] C. E. Leiserson and T. B. Schardl. A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers). In Proceedings of the Twenty-second Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '10, pages 303–314, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0079-7. doi: 10.1145/1810479.1810534. URL http://doi.acm.org/10.1145/1810479.1810534.

[59] A. Leist and A. Gilman. A comparative analysis of parallel programming models for c++. In In The Ninth International Multi-Conference on Computing in the Global Information Technology, ICCGI 2014, pages 121–127, March 2014. ISBN Arno Leist Andrew Gilman.

[60] J. Leskovec and A. Krevl. Snap for c++. http://snap.stanford.edu/snap/download.html, June 2014.

[61] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[62] J. Leskovec and R. Sosič. Snap: A general-purpose network analysis and graph-mining library. ACM Transactions on Intelligent Systems and Technology (TIST), 8(1):1, 2016.

[63] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. CoRR, abs/0810.1355, 2008. URL http://arxiv.org/abs/0810.1355.

[64] H. Li. Centrality analysis of online social network big data. In 2018 IEEE 3rd International Conference on Big Data Analysis (ICBDA), pages 38–42, March 2018. doi: 10.1109/ICBDA.2018.8367648.

[65] Q. Li and W. Wei. A parallel single-source shortest path algorithm based on bucket structure. In 2013 25th Chinese Control and Decision Conference (CCDC), pages 3445–3450, May 2013. doi: 10.1109/CCDC.2013.6561544.

[66] LiveStats. Internet live stats. http://www.internetlivestats.com/.

[67] A. Louni and K. P. Subbalakshmi. Who spread that rumor: Finding the source of information in large online social networks with probabilistically varying internode relationship strengths. IEEE Transactions on Computational Social Systems, 5(2): 335–343, June 2018. ISSN 2329-924X. doi: 10.1109/TCSS.2018.2801310.

[68] A. Mahmoody, C. E. Tsourakakis, and E. Upfal. Scalable betweenness centrality maximization via sampling. In Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, pages 1765–1773, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939869. URL http://doi.acm.org/10.1145/2939672.2939869.

[69] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A system for large-scale graph processing. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0032-2. doi: 10.1145/1807167.1807184. URL http://doi.acm.org/10.1145/1807167.1807184.

[70] A. McLaughlin and D. A. Bader. Accelerating gpu betweenness centrality. Communications of the ACM, 61(8):85–92, July 2018. ISSN 0001-0782. doi: 10.1145/3230485. URL http://doi.acm.org/10.1145/3230485.

[71] L. Mertz. What can big data tell us about health?: Finding gold through data mining. IEEE Pulse, 7(5):40–44, Sep. 2016. doi: 10.1109/MPUL.2016.2592242.

[72] R. T. Michael T. Goodrich and M. H. Goldwasser. Data Structures and Algorithms in Java. Wiley, 6th edition, 2014. ISBN 1118771338.

[73] Microsoft. Processes and threads. https://docs.microsoft.com/en-us/windows/desktop/procthread/processes-and-threads.

[74] D. K. Mishra, X.-S. Yang, and A. Unal. Data Science and Big Data Analytics. Springer, 1st edition, 2018.

[75] J. Mondal and A. Deshpande. Managing large dynamic graphs efficiently. In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, SIGMOD '12, pages 145–156, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1247-9. doi: 10.1145/2213836.2213854. URL http://doi.acm.org/10.1145/2213836.2213854.

[76] E. F. Moore. The shortest path through a maze. In International Symposium on the Theory of Switching, page 285–92, 1959.

[77] F. Moradi. Improving Community Detection Methods for Network Data Analysis. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, Sweden, 2014.

[78] A. Mukkara, N. Beckmann, M. Abeydeera, X. Ma, and D. Sanchez. Exploiting locality in graph analytics through hardware-accelerated traversal scheduling. In 51st Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2018, Fukuoka, Japan, October 20-24, 2018, pages 1–14, 2018. doi: 10.1109/MICRO.2018.00010.

[79] E. R. Nathan. Analyzing Centrality Indices in Complex Networks: an Approach Using Fuzzy Aggregation Operators. PhD thesis, Department of Computer Science, Technische Universit at Kaiserslautern, 2018.

[80] W. Nawaz, K. Khan, and Y. Lee. Core analysis for efficient shortest path traversal queries in social graphs. In 2014 IEEE Fourth International Conference on Big Data and Cloud Computing, pages 363–370, Dec 2014. doi: 10.1109/BDCloud.2014.11.

[81] NetworkX. Software for complex networks. http://networkx.github.io.

[82] K. Okamoto, W. Chen, and X.-Y. Li. Ranking of closeness centrality for large-scale social networks. In Proceedings of the International Workshop on Frontiers in Algorithmics, FAW'2008, pages 186–195, 2008.

[83] H. Ortega-Arranz, D. R. Llanos, and A. Gonzalez-Escribano. The Shortest-Path Problem: Analysis and Comparison of Methods. Morgan & Claypool, 2014. ISBN 1627055398.

[84] D. Ortiz-Arroyo. Discovering Sets of Key Players in Social Networks, chapter 2, pages 201–213. Springer, London, 2010. ISBN 978-1-84882-228-3.

[85] E. Otte and R. Rousseau. Social network analysis: a powerful strategy, also for the information sciences. Journal of Information Science, 28(6):441–453, 2002. doi: 10.1177/016555150202800601. URL https://doi.org/10.1177/016555150202800601.

[86] M. Park, S. Lee, O. Kwon, and A. Seuret. Closeness-centrality-based synchronization criteria for complex dynamical networks with interval time-varying coupling delays. IEEE Transactions on Cybernetics, 48(7):2192–2202, July 2018. ISSN 2168-2267. doi: 10.1109/TCYB.2017.2729164.

[87] S. R. Proulx, D. E. Promislow, and P. C. Phillips. Network thinking in ecology and evolution. Trends in Ecology & Evolution, 20:345–353, 2005. ISSN 00380431. doi: 10.1016/j.tree.2005.04.004.

[88] R. Puzis, Y. Elovici, P. Zilberman, S. Dolev, and U. Brandes. Topology manipulations for speeding betweenness centrality computation. Journal of Complex Networks, 3(1): 84–112, March 2015. ISSN 2051-1310. doi: 10.1093/comnet/cnu015.

[89] M. Riondato and E. M. Kornaropoulos. Fast approximation of betweenness centrality through sampling. Data Mining and Knowledge Discovery, 30:438–475, 2016. ISSN 1384-5810. doi: 10.1007/s10618-015-0423-0.

[90] A. Roy. Memory hierarchy sensitive graph layout. Computing Research Repository-CoRR, abs/1203.5675, 2012. URL http://arxiv.org/abs/1203.5675.

[91] A. E. Sariyüce, K. Kaya, E. Saule, and U. V. Çatalyürek. Graph manipulations for fast centrality computation. ACM Trans. Knowl. Discov. Data, 11(3):26:1–26:25, Mar. 2017. ISSN 1556-4681. doi: 10.1145/3022668. URL http://doi.acm.org/10.1145/3022668.

[92] G. Scano, M. Huguet, and S. U. Ngueveu. Adaptations of k-shortest path algorithms for transportation networks. In 2015 International Conference on Industrial Engineering and Systems Management (IESM), pages 663–669, Oct 2015. doi: 10.1109/IESM.2015.7380229.

[93] S. E. Schaeffer. Graph clustering. Computer Science Review, 1(1):27 – 64, 2007. ISSN 1574-0137. doi: https://doi.org/10.1016/j.cosrev.2007.05.001. URL http://www.sciencedirect.com/science/article/pii/S1574013707000020.

[94] J. Scott and P. J. Carrington. The SAGE Handbook of Social Network Analysis. Web Information Systems Engineering and Internet Technologies. SAGE Publications Ltd., London, 2014. ISBN 9781847873958. doi: 10.4135/9781446294413.

[95] R. Sedgewick and P. Flajolet. An Introduction to the Analysis of Algorithms. Addison-Wesley Professional, 2nd edition, 2013. ISBN 032190575X.

[96] S. Sherif and E. Pardede. Graph Data Management: Techniques and Applications. IGI Global, 1st edition, 2012.

[97] SigMod. The acm sigmod programming contest. `http://dsg.uwaterloo.ca/sigmod16contest/`.

[98] G. M. Slota. Irregular Graph Algorithms On Modern Multicore, Manycore, And Distributed Processing Systems. PhD thesis, College of Engineering, The Pennsylvania State University, 2016.

[99] C. L. STAUDT, A. SAZONOVS, and H. MEYERHENKE. Networkit: A tool suite for large-scale complex network analysis. Network Science, 4(4):508–530, 2016. doi: 10.1017/nws.2016.20.

[100] L. Takac and M. Zabovsky. Data analysis in public social networks. In Proceedings of the International Scientific Conference & International Workshop Present Day Trends of Innovations, pages 272–278, May 2012. ISBN 978-3-319-27260-3.

[101] F. W. Takes and E. M. Heemskerk. Centrality in the global network of corporate control. Social Network Analysis and Mining, 6, 2016. doi: 10.1007/s13278-016-0402-5.

[102] R. Trobec, B. Slivnik, P. Bulić, and B. Robič. Introduction to Parallel Computing. From Algorithms to Programming on State-of-the-Art Platforms. Springer, 2018. ISBN 978-3-319-98833-7.

[103] R. J. Trudeau. Introduction to Graph Theory. Dover Publications, 2nd edition, 1994.

[104] L. H. U, H. J. Zhao, M. L. Yiu, Y. Li, and Z. Gong. Towards online shortest path computation. IEEE Transactions on Knowledge and Data Engineering, 26(4):1012–1025, April 2014. ISSN 1041-4347. doi: 10.1109/TKDE.2013.176.

[105] F. Vella, M. Bernaschi, and G. Carbone. Dynamic merging of frontiers for accelerating the evaluation of betweenness centrality. J. Exp. Algorithmics, 23:1.4:1–1.4:19, Mar. 2018. ISSN 1084-6654. doi: 10.1145/3182656. URL `http://doi.acm.org/10.1145/3182656`.

[106] S. Wasserman and K. Faust. Social Network Analysis: Methods and Applications (Structural Analysis in the Social Sciences). Cambridge University Press, 1st edition, 1994. ISBN 0521382696.

[107] J. Wei, K. Chen, Y. Zhou, Q. Zhou, and J. He. Benchmarking of distributed computing engines spark and graphlab for big data analytics. In 2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService), pages 10–13, March 2016.

[108] R. J. Wilson. Introduction to Graph Theory. Pearson Publisher, 5th edition, 2010.

[109] G. Xu, Y. Zhang, and L. Li. Web Mining and Social Networking: Techniques and Applications. Web Information Systems Engineering and Internet Technologies. Springer US, 1st edition, 2011. ISBN 1441977341.

[110] S. A. Yahia, M. Benedikt, L. V. S. Lakshmanan, and J. Stoyanovich. Efficient network aware search in collaborative tagging sites. Proc. VLDB Endow., 1(1):710–721, Aug. 2008. ISSN 2150-8097. doi: 10.14778/1453856.1453934. URL http://dx.doi.org/10.14778/1453856.1453934.

[111] N. Yakovets. Query Processing On Attributed Graphs. PhD thesis, University of Pittsburgh, 2016.

[112] J. Yang and Y. Chen. Fast computing betweenness centrality with virtual nodes on large sparse networks. PLOS ONE, 6(7):1–5, 07 2011. doi: 10.1371/journal.pone.0022557. URL https://doi.org/10.1371/journal.pone.0022557.

[113] Y. Zhang, J. Tang, Z. Yang, J. Pei, and P. S. Yu. Cosnet: Connecting heterogeneous social networks with local and global consistency. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '15, pages 1485–1494, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3664-2. doi: 10.1145/2783258.2783268. URL http://doi.acm.org/10.1145/2783258.2783268.

[114] Y. Zhang, V. Kiriansky, C. Mendis, S. Amarasinghe, and M. Zaharia. Making caches work for graph analytics. In 2017 IEEE International Conference on Big Data (Big Data), pages 293–302, 2017. doi: 10.1109/BigData.2017.8257937.