

ĐẠI HỌC THĂNG LONG
KHOA HỌC MÁY TÍNH



KHÓA LUẬN TỐT NGHIỆP
PHÁT HIỆN CỘNG ĐỒNG TRONG
MẠNG TƯƠNG TÁC KÍCH THƯỚC LỚN

GIẢNG VIÊN HƯỚNG DẪN:
TS. TRẦN VĨNH ĐỨC

SINH VIÊN THỰC HIỆN:
NGUYỄN ĐỨC THẮNG - A22852

HÀ NỘI - 2017

LỜI CẢM ƠN

Lời đầu tiên, tôi xin gửi lời cảm ơn trân trọng và sâu sắc tới các thầy cô thuộc Bộ môn Tin học - Khoa Toán tin trường Đại học Thăng Long, đã tận tâm truyền đạt những kiến thức quý báu trong quá trình học tập và thực hiện khóa luận này.

Đặc biệt, tôi xin gửi lời cảm ơn chân thành và sâu sắc tới TS. Trần Vĩnh Đức, người đã trực tiếp hướng dẫn tận tình và đóng góp những ý kiến quý báu và Ths. Trần Tuấn Toàn đã tạo điều kiện sử dụng phòng máy trong quá trình thực nghiệm khóa luận này.

Cuối cùng, tôi xin được gửi lời cảm ơn đến gia đình, người thân và bạn bè của tôi, những người đã ở bên, khuyến khích và động viên trong cuộc sống, học tập.

Trong quá trình thực hiện khóa luận của mình, tôi đã có gắng hết sức để tìm hiểu và hoàn thiện một cách tốt nhất. Nhưng với kiến thức và sự hiểu biết còn hạn chế, khóa luận này sẽ không tránh khỏi những thiếu sót, kính mong nhận được những góp ý của các thầy cô, các bạn và những người quan tâm đến khóa luận này.

Tôi xin chân thành cảm ơn!

Sinh viên thực hiện

Nguyễn Đức Thắng

LỜI CAM ĐOAN

Tôi xin cam đoan đề tài phát hiện cộng đồng trong mạng xã hội lớn, thực hiện dựa trên mô hình BigCLAM và các thuật toán học máy được trình bày trong khóa luận là do tôi thực hiện dưới sự hướng dẫn của TS. Trần Vĩnh Đức.

Tất cả các bài báo, tài liệu, công cụ phần mềm của các tác giả khác được sử dụng trong khóa luận này đều được chỉ dẫn tường minh về nguồn và tác giả trong danh sách tài liệu tham khảo.

Hà Nội, ngày 05 tháng 07 năm 2017

Sinh viên thực hiện

Nguyễn Đức Thắng

MỞ ĐẦU

Với sự phát triển nhanh chóng của các cộng đồng mạng kết nối như mạng xã hội Facebook, Twitter, Youtube,... hay mạng thương mại điện tử như Amazon, Netflix, Lazada,... việc phân tích các mạng này đóng vai trò ngày càng quan trọng, thời sự và được sự quan tâm nghiên cứu thuộc nhiều lĩnh vực như xã hội học, kinh tế, khoa học máy tính,... Trong đó bài toán phát hiện cộng đồng trong các mạng tương tác đang là một trong những nội dung nghiên cứu dành được nhiều sự quan tâm từ phía các nhà khoa học. Các nhà khoa học đã đề xuất rất nhiều các phương pháp phát hiện cộng đồng trong mạng tương tác, đặc biệt là các mạng có tính chồng chéo, trong số đó mô hình BigCLAM (Cluster Affiliation Model for Big Networks) được Yang.J và Leskovec.L đề xuất năm 2013 được chứng minh là khá nổi trội trong việc phát hiện cộng đồng chồng chéo trong mạng có kích thước lớn.

Dề tài này sẽ đi sâu vào nghiên cứu mô hình BigCLAM và đề xuất một vài phương pháp huấn luyện cũng như phương pháp tối ưu tốc độ huấn luyện. Trong quá trình thực nghiệm khóa luận, phương pháp được cài đặt trên hệ thống tính toán phân tán và sử dụng mạng có kích thước lớn trên trang SNAP¹ của Đại học Stanford như Facebook, Amazon, Youtube, ...

Nội dung của khóa luận gồm 5 chương:

Chương 1: Tổng quan về mạng và bài toán phát hiện cộng đồng chồng chéo. Tại chương này của khóa luận trình một cách khái quát về mô hình mạng tương tác trong thực tế và các bài toán liên quan đặc biệt là bài toán phát hiện cộng đồng. Từ đó thiết lập động lực và mục tiêu của khóa luận này.

Chương 2: Cơ sở lý thuyết. Trong chương này sẽ trình bày chi tiết các khái niệm cơ sở được sử dụng trong khóa luận.

Chương 3: Phương pháp phát hiện cộng đồng sử dụng mô hình Big-CLAM. Chương này sẽ trình bày một cách chi tiết về mô hình BigCLAM sử dụng phương pháp cực đại hàm khả dĩ để giải quyết bài toán phát hiện cộng đồng.

Chương 4: Cài đặt phương pháp phát hiện cộng đồng sử dụng mô hình BigCLAM trên hệ thống phân tán. Tại chương này sẽ giới thiệu mô hình tính

¹<http://snap.stanford.edu>

toán phân tán Apache Hadoop và Apache Spark trong xử lý dữ liệu lớn. Đồng thời đề xuất mô hình cài đặt BigCLAM mới chạy trên hệ thống tính toán phân tán.

Chương 5: Kết luận và hướng phát triển. Dưa ra những kết quả của khóa luận và trình bày phương hướng phát triển sau khóa luận.

Mục lục

| | |
|---|-------------|
| Mục lục | v |
| Danh sách hình vẽ | vii |
| Danh sách bảng | viii |
| Danh mục ký hiệu và từ viết tắt | ix |
| 1 Tổng quan về mạng tương tác và bài toán phát hiện cộng đồng | 1 |
| 1.1 Tổng quan về mạng tương tác | 1 |
| 1.2 Bài toán phát hiện cộng đồng | 3 |
| 1.3 Một số nghiên cứu | 5 |
| 1.3.1 Phương pháp Girvan-Newman | 5 |
| 1.3.2 Phương pháp CONGA | 6 |
| 1.3.3 Affiliation Graph Model (AGM) | 7 |
| 1.3.4 Non-negative Matrix Factorization | 7 |
| 2 Cơ sở lý thuyết | 9 |
| 2.1 Vector và ma trận | 9 |
| 2.2 Lý thuyết đồ thị | 9 |
| 2.3 Một số phương pháp ước lượng và tối ưu tham số | 11 |
| 2.3.1 Ước lượng bằng cực đại khả dĩ | 11 |
| 2.3.2 Các phương pháp tối ưu hàm lồi | 12 |
| 3 Phương pháp phát hiện cộng đồng dựa trên mô hình BigCLAM | 16 |
| 3.1 Mô hình BigCLAM | 16 |
| 3.2 Phát hiện cộng đồng sử dụng mô hình BigCLAM | 17 |
| 3.2.1 Cực đại hàm khả dĩ ma trận trọng số liên kết của đồ thị | 18 |
| 3.2.2 Khởi tạo ma trận trọng số liên kết cộng đồng | 19 |
| 3.2.3 Xác định các liên kết cộng đồng | 19 |
| 3.2.4 Xác định số cộng đồng | 20 |

| | | |
|----------|---|-----------|
| 3.3 | Dánh giá hiệu năng | 20 |
| 4 | Cài đặt phương pháp phát hiện cộng đồng sử dụng mô hình BigCLAM trên hệ thống phân tán | 22 |
| 4.1 | Tổng quan hệ tính toán phân tán | 22 |
| 4.1.1 | Giới thiệu Apache Hadoop & Spark | 22 |
| 4.1.2 | Tính toán song song và phân tán với RDD | 27 |
| 4.2 | Thực nghiệm | 29 |
| 5 | Kết luận và hướng phát triển | 32 |
| 5.1 | Kết luận | 32 |
| 5.2 | Hướng phát triển | 33 |
| | Tài liệu tham khảo | 34 |
| | Phụ lục A: | |
| | Cài đặt và triển khai ứng dụng trên Apache Hadoop & Spark | 36 |
| A.1 | Cài đặt | 36 |
| A.1.1 | Apache Hadoop & Spark | 37 |
| A.1.2 | Cấu hình Hadoop | 39 |
| A.1.3 | Cấu hình Spark | 40 |
| A.2 | Tập lệnh | 42 |
| A.2.1 | Danh sách tập lệnh làm việc trên Hadoop | 42 |
| A.2.2 | Danh sách tập lệnh làm việc trên Spark | 43 |

Danh sách hình vẽ

| | | |
|-----|--|----|
| 1.1 | Dữ liệu được sinh ra sau 60s | 1 |
| 1.2 | Một mạng tương tác (đồ thị) đơn giản với ba cộng đồng. | 4 |
| 1.3 | Mô tả các cộng đồng trong mạng mạng cá nhân (egonetwork) của một sinh viên Đại học Stanford | 4 |
| 1.4 | Các loại cộng đồng mà trong mạng tương tác có thể có | 5 |
| 1.5 | Minh họa phương pháp Girvan-Newman | 6 |
| 1.6 | Minh họa mô hình AGM | 7 |
| 2.1 | Đồ thị hai phần | 11 |
| 2.2 | Hàm lồi. | 13 |
| 2.3 | Hàm lồi (trái) và hàm không lồi (phải) | 13 |
| 2.4 | Sự ảnh hưởng của α đến quá trình huấn luyện. | 14 |
| 3.1 | Mô hình cộng đồng BigCLAM | 17 |
| 3.2 | Ma trận trọng số kết nối cộng đồng | 17 |
| 3.3 | So sánh hiệu năng của các mô hình với BigCLAM | 20 |
| 3.4 | Biểu đồ so sánh độ chính xác các mô hình trên các mạng biết trước cộng đồng | 21 |
| 4.1 | Kiến trúc và hệ sinh thái của Apache Hadoop | 23 |
| 4.2 | Kiến trúc và hệ sinh thái Apache Spark | 24 |
| 4.3 | So sánh tốc độ sắp xếp 1TB dữ liệu của Apache Hadoop và Apache Spark . | 25 |
| 4.4 | Biểu đồ thể hiện sự quan tâm của cộng đồng đến Apache Spark | 26 |
| 4.5 | Biểu đồ thể hiện sự so sánh mức độ quan tâm của các hệ sinh thái Apache Spark | 26 |
| 4.6 | Mô hình kết hợp | 27 |
| 4.7 | Kết quả thực nghiệm | 30 |
| 4.8 | Mô phỏng phát hiện cộng đồng trên mạng facebook 4.4 | 31 |
| 4.9 | Mô phỏng phát hiện cộng đồng trên mạng internet 4.4 | 31 |

Danh sách bảng

| | | |
|-----|--|----|
| 3.1 | Mô tả các mạng sử dụng để thực nghiệm | 21 |
| 4.1 | Danh sách các thao tác transformation | 28 |
| 4.2 | Danh sách các thao tác action | 29 |
| 4.3 | Thông số hệ thống sử dụng trong thực nghiệm | 30 |
| 4.4 | Mô tả các mạng sử dụng để thực nghiệm | 30 |
| 1 | Các địa chỉ IP trong cụm | 36 |
| 2 | Các phiên bản cài đặt | 38 |
| 3 | Danh sách tập lệnh cơ bản làm việc trên Hadoop | 42 |
| 4 | Danh sách tập lệnh cơ bản làm việc trên Spark | 43 |

Danh mục ký hiệu và từ viết tắt

Dưới đây là danh sách các ký hiệu và từ viết tắt đã được sử dụng trong đề tài này.

| | |
|----------------------------------|---|
| α | Tham số tốc huân luyện |
| $\mathcal{A}(\mathcal{S})$ | Tổng số bậc bên trong mỗi đồ thị con S |
| \mathcal{C} | Tập cộng đồng |
| \mathcal{E} | Tập cạnh |
| K | Số cộng đồng |
| $\mathcal{N}(u)$ | Đỉnh lân cận của đỉnh u |
| \mathcal{V} | Tập đỉnh |
| $\nabla_{\theta}f(\theta)$ | Đạo hàm của hàm số $f(\theta)$ theo θ |
| $\varphi(\mathcal{S})$ | Độ dẫn của một đồ thị con \mathcal{S} |
| A | Ma trận kè của đồ thị $G(\mathcal{V}, \mathcal{E})$ |
| $B(\mathcal{V}, \mathcal{C}, M)$ | Đồ thị hai phần thể hiện mô hình AGM |
| $dom f$ | Tập xác định của hàm số $f(.)$ |
| F | Ma trận trọng số kết nối cộng đồng |
| F_u | Vector trọng số kết nối cộng đồng |
| $G(\mathcal{V}, \mathcal{E})$ | Đồ thị có \mathcal{V} đỉnh và \mathcal{E} cạnh |
| $J(\theta)$ | Hàm chi phí |
| $L(.)$ | Hàm khả dĩ |

| | |
|------------|---|
| $l(\cdot)$ | Logarit tự nhiên của hàm khả dĩ |
| M | Cạnh liên kết các đỉnh với các cộng đồng |
| $N(u)$ | Vùng lân cận của đỉnh u |
| $p(u, v)$ | Xác suất tạo ra cạnh $(u, v) \in \mathcal{E}$ |
| p_A | Xác suất của một cộng đồng |
| $MBSGD$ | Mini-batch Stochastic Gradient descent |
| SGD | Stochastic Gradient descent |
| AGM | Affiliation graph model |
| BigCLAM | Cluster Affiliation Model for Big Networks |
| CONGA | Cluster Overlap Newman Girvan Algorithm |
| GD | Gradient descent |
| GFS | Google File System |
| HDFS | Hadoop Distributed File System |
| MLE | Maximum-Likelihood Estimation |
| NMF | Non-negative Ma trận Factorization |
| SNAP | Stanford Network Analysis Project |

Chương 1

Tổng quan về mạng tương tác và bài toán phát hiện cộng đồng

1.1 Tổng quan về mạng tương tác

Big Data là một thuật ngữ dùng để chỉ một tập hợp dữ liệu có kích thước lớn và phức tạp mà không thể xử lý trên những công cụ xử lý dữ liệu truyền thống. Thống kê cho thấy, trong hai năm qua khối dữ liệu trên toàn cầu đã chiếm đến 90% lượng dữ liệu số được tạo ra kể từ khi công nghệ số hóa ra đời. Theo dự đoán của các chuyên gia dữ liệu nhận định rằng tốc độ tăng trưởng của dữ liệu là 62% mỗi năm dự đoán đến năm 2020 chúng ta sẽ có 40+ exabytes.



Hình 1.1: Dữ liệu được sinh ra sau 60s.

Nhìn vào hình 1.1 số liệu (2017)¹ cho thấy trung bình cứ 60s mạng xã hội Facebook có 336, 927 status updates, 172, 150 messages, 57, 477 shared links, 115, 023 friends requests,... Tương tự như Amazon, Google+, Youtube, Tweeter,... đều là những mạng tương tác vô cùng lớn là nơi lưu trữ các thông tin, quan điểm, tính cách, các tương tác giữa các cá thể,... Những thông tin này tạo thành đám mây tri thức vô cùng lớn mà chúng ta cần phải khai thác và đưa vào sử dụng. Có thể thấy trong kỷ nguyên số này thì dữ liệu chính là nguồn nhiên liệu vô cùng quan trọng của hầu hết các lĩnh vực. Việt Nam trong thời gian qua cũng đã và đang bắt đầu tiến đến cuộc cách mạng công nghiệp lần thứ tư thúc đẩy quá trình sản xuất thông minh dựa trên các thành tựu về công nghệ thông tin, công nghệ sinh học và công nghệ nano trong đó những bài toán về khai thác và xử lý dữ liệu được đặc biệt quan tâm. Theo lời của GS. Hồ Tú Bảo đã nói "Tới đây hầu hết các công ty đều phải sống dựa vào dữ liệu, ai có nhiều dữ liệu hơn và khai phá được nhiều hơn thì người đó sẽ thắng".

Nhìn nhận lại chúng ta thấy mạng tương tác xuất hiện trong nhiều lĩnh vực như: xã hội học, công nghệ thông tin, khoa học hành vi, toán học, thống kê, y học và nhiều lĩnh vực khác. Mạng tương tác hiện nay được chia ra thành các dạng khác nhau như mạng hiện - ẩn, tĩnh - động, ngoại tuyến - trực tuyến. Chủ yếu dữ liệu mạng tương tác được phân thành hai loại chính như sau:

- Nội dung: Chính là các thông tin được lưu trữ trên mạng. Nội dung có thể được hiểu dưới nhiều góc độ khác nhau trong nhiều lĩnh vực như mạng xã hội là các thông tin, trạng thái, bình luận của người sử dụng hay trong mạng sinh học gen chính là các chuỗi ADN thông tin. Do đó, kỹ thuật xử lý thường được sử dụng là các phương pháp xử lý văn bản. Tuy nhiên, do tính chất biến động và độ lớn của mạng xã hội hay tính không đầy đủ các thông tin được chia sẻ từ người dùng nên dữ liệu văn bản của mạng xã hội khác với các dữ liệu văn bản truyền thống trước đây. Những bài toán sử dụng dữ liệu này là: phân tích quan điểm người dùng trên mạng xã hội, tìm kiếm chủ đề nổi bật trên mạng xã hội,...
- Cấu trúc: Chính là sự tương tác giữa các cá thể trong mạng. Ví dụ trong mạng xã hội là quan hệ kết bạn, like, share,... trong mạng thương mại là quan hệ mua hàng, bán hàng,... Cụ thể mạng tương tác được định nghĩa như là một mô hình đồ thị được cấu tạo bởi các đỉnh và các cạnh. Các đỉnh là tập các đối tượng, các cạnh là tập các tương tác giữa các cá thể. Các bài toán sử dụng dữ liệu này là: dự đoán liên kết trong mạng xã hội (được chia thành bốn bài toán con: dự đoán sự tồn tại của liên kết, dự đoán loại liên kết, dự đoán số liên kết, dự đoán số trọng số liên kết), gom nhóm hoặc phân lớp cộng đồng các cá thể trong mạng (đây chính là bài toán sẽ được đề cập đến trong khóa luận này),...

¹<http://www.coupoify.com/social-media-in-realtime>

1.2 Bài toán phát hiện cộng đồng

Theo định nghĩa của trên trang Oxford English Dictionary (2017)¹ thì từ "Communities: A group of people living in the same place or having a particular characteristic in common.". Có thể hiểu, cộng đồng là một nhóm các cá thể trong mạng có những tính chất tương tự nhau và cùng đóng một vai trò trong mạng. Chúng ta có ánh xạ khái niệm cộng đồng trong các mạng tương tác thường thấy như:

- Mạng xã hội là một trong những ví dụ điển hình của đồ thị các cộng đồng. Con người có xu hướng kết nối với nhau lại với nhau, hình thành các nhóm (cộng đồng) trong cùng một môi trường làm việc, gia đình, bạn bè, sở thích,...
- Nghiên cứu y sinh học: Tương tác giữa các protein trong cùng một mô-đun (cộng đồng) sẽ có tính nổi trội hơn.
- Hệ khuyến nghị: Khách hàng có thể mua các sản phẩm từ nhóm các sản phẩm có liên quan đến nhau.
- ...

Như vậy nếu các protein, con người, hàng hóa là các tác nhân trong mạng thì các mô-đun, nhóm người dùng hay nhóm sản phẩm chính là các cộng đồng. Và nếu chúng ta phát hiện ra những cộng đồng tiềm năng này cho phép tạo ra rất nhiều giá trị như:

- Việc hiểu và phát hiện ra các cộng đồng khách hàng giúp cho phép xây dựng chiến lược kinh doanh tốt nhất, đem lại hiệu quả cao trong kinh doanh.
- Trong y sinh học, việc tìm ra các cộng đồng gen, ADN, ... giúp cho chúng ta hiểu sâu hơn về cấu trúc sinh học và từ đó đưa ra các phương pháp trị liệu tốt nhất với từng loại bệnh.
- Hiểu và tìm ra các cộng đồng trên mạng xã hội, giúp cho các nhà phân tích có thể đưa ra được những chính sách xã hội, đầu tư hay đưa ra được những dự đoán mang tính xã hội. Ví dụ trong đợt vận động tranh cử nước tổng thống Mỹ, các chuyên gia tin học của tổng thống Donald Trump đã tiến hành phân tích mạng xã hội để dự đoán khả năng trúng cử đồng thời đưa ra những biện pháp đến với từng đối tượng để tăng cơ hội đắc cử.
- ...

Hình1.2 là một ví dụ đơn giản của một mạng tương tác với ba cộng đồng và ta có phát biểu bài toán phát hiện cộng đồng trong toán học như sau:

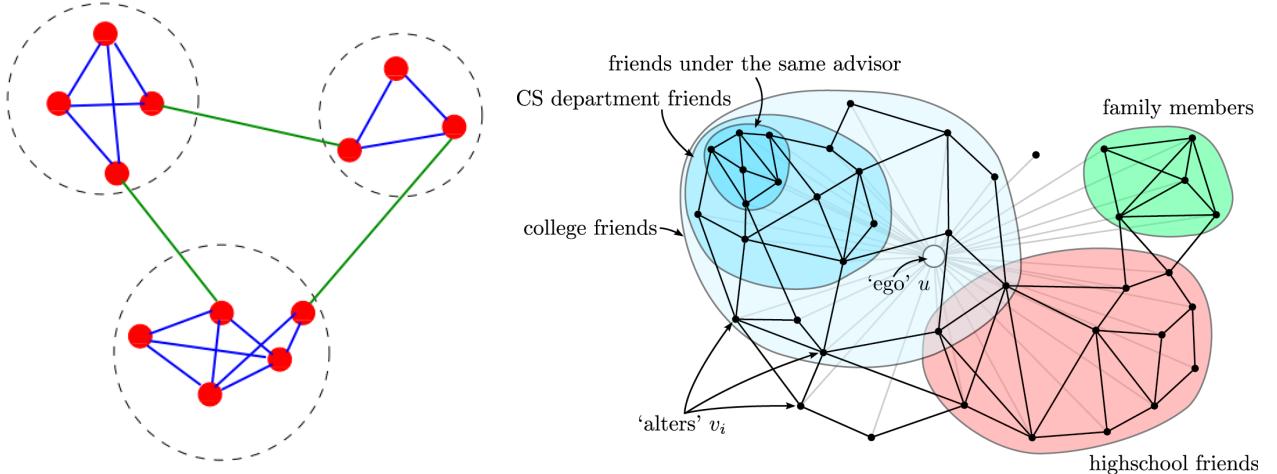
¹<https://en.oxforddictionaries.com/definition/community>

Bài toán 1 (*Phát hiện cộng đồng trong mạng*)

Cho $G(\mathcal{V}, \mathcal{E})$ là một đồ thị với tập cạnh \mathcal{E} và tập đỉnh \mathcal{V} , chúng ta cần khám phá và phát hiện các cộng đồng C_1, C_2, \dots, C_K với K là số lượng cộng đồng. Trong đó mỗi cộng đồng $C_i \subseteq \mathcal{V}, \forall i \in [0, K]$.

Nếu ta thêm một ràng buộc trong bài toán các C_i là riêng biệt không có sự chồng chéo giữa chúng tức $C_i \cap C_j = \emptyset, \forall i, j \in [0, K]$, thì ta sẽ có một khái niệm mới cho bài toán 1 là phát hiện cộng đồng không có sự chồng chéo (non-overlapping community detection). Ngược lại, bài toán khóa luận này đề cập đến chính là phát hiện cộng đồng ở đó C_i có khả năng chồng chéo (overlapping community detection).

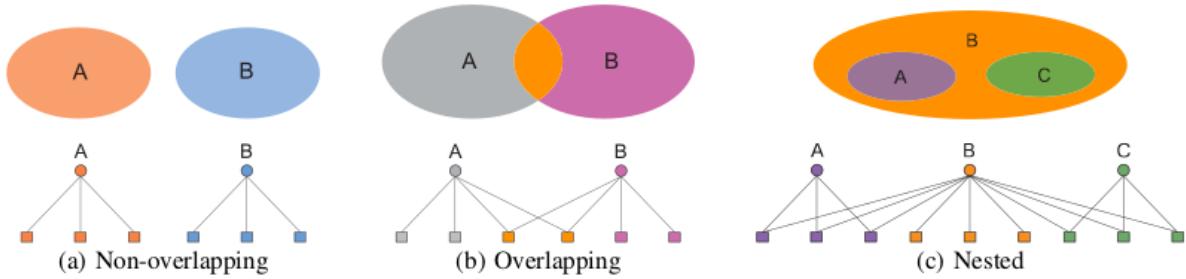
Hay nói cách khác, phát hiện cộng đồng mạng hay có thể hiểu là việc phân cụm các đỉnh trong một mạng sử dụng cấu trúc mạng kết nối. Và K ở đây chính là số cộng đồng được ước lượng là tốt nhất.



Hình 1.2: Một mạng tương tác (đồ thị) Hình 1.3: Mô tả các cộng đồng trong mạng cá nhân (egonetwork) của một sinh viên Đại học Stanford

Thực tế hầu như các cộng đồng đều giàu tính chồng chéo hoặc là tập con của một cộng đồng khác. Có thể thấy rõ trong mạng xã hội, hình 1.3 là một kết quả của McAuley and Leskovec [2012] [12] cho thấy trong mạng cá nhân của một sinh viên có rất nhiều các cộng đồng khác nhau như family members, highschoold fiends, college friends,... Và những cộng đồng này đều có tính chồng chéo và có cả những cộng đồng là con của một cộng đồng khác. Vì vậy bài toán phát hiện cộng đồng trong các mạng có đầy đủ cả ba loại cộng đồng như hình 1.4 là một bài toán khó trong khi sự bùng nổ về kích thước và độ phức tạp của các mạng tăng theo từng phút làm cho bài toán này càng trở nên khó khăn.

Với tiềm năng đầy giá trị của bài toán cũng như những thách thức của bài toán đặt ra, chính là động lực để tôi chọn và thực hiện khóa luận này. Đề tài giải quyết bài toán phát hiện cộng đồng trong các mạng tương tác có kích thước lớn có tính không chồng chéo, chồng chéo



Hình 1.4: Các loại cộng đồng mà trong mạng tương tác có thể có

và lồng (chứa) nhau trong thời gian cho phép. Trong các chương tiếp theo của khóa luận, tôi sẽ trình bày chi tiết giải quyết bài toán dựa trên mô hình BigCLAM được đề xuất bởi Yang and Leskovec [2013] [16] kết hợp với một số phương pháp huấn luyện chạy trên hệ thống tính toán lưới.

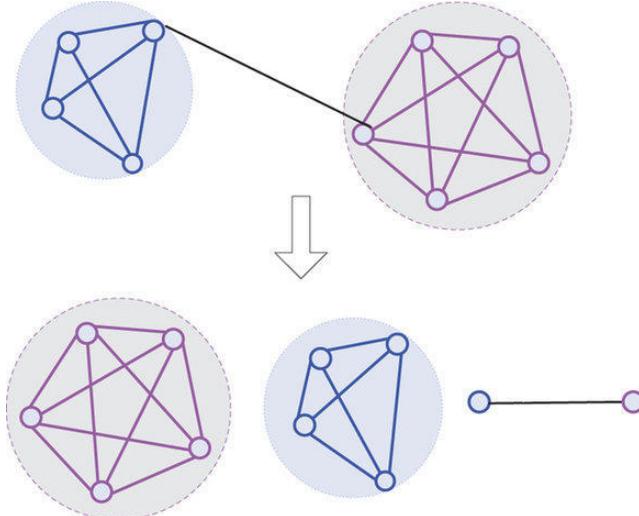
1.3 Một số nghiên cứu

Bài toán phát hiện cộng đồng nhận được sự quan tâm từ rất sớm và có rất nhiều phương pháp giải quyết bài toán 1 có kết quả khá tốt. Tuy nhiên mỗi phương pháp đều có những khuyết điểm, thường không phù hợp các mạng có kích thước lớn hiện nay.

1.3.1 Phương pháp Girvan-Newman

Phát hiện cộng đồng dựa trên việc tìm cạnh nối (cạnh cầu) giữa các cộng đồng. Thuật toán này dựa trên quan niệm cho rằng khi các cộng đồng được gắn kết với nhau thì đường đi giữa các cộng đồng này đến cộng đồng khác sẽ đi qua các cạnh nối giữa chúng với tần suất cao. Điểm quan trọng nhất của phương pháp là xây dựng cộng đồng bằng cách loại bỏ dần dần các cạnh nối (cạnh cầu) từ đồ thị ban đầu. Phương pháp điển hình nhất trong sử dụng tính chất này là phương pháp Girvan-Newman của Newman [2004] [13]. Về cơ bản phương pháp Girvan-Newman được thực hiện theo các bước sau:

1. Tính độ đo trung gian cho tất cả các cạnh trong mạng;
2. Loại bỏ các cạnh có độ trung gian cao nhất;
3. Tính lại độ trung gian cho tất cả các cạnh bị ảnh hưởng theo các cạnh đã loại bỏ;
4. Lặp lại từ bước 2 cho đến khi không còn các cạnh trung gian ta sẽ có các cộng đồng.



Hình 1.5: Minh họa phương pháp Girvan-Newman

Thuật toán Girvan-Newman khá đơn giản, dễ hiểu và có những kết quả tương đối tốt trong nhiều trường hợp, mặc dù vậy nó vẫn gặp phải một số nhược điểm sau:

1. Việc tính lại toàn bộ độ trung gian sau mỗi lần loại bỏ các cạnh có độ trung gian cao nhất. Như vậy khiến cho độ phức tạp của thuật toán rất lớn $O(m \times n)$ cho mỗi lần lặp với m là số cạnh và n là số đỉnh. Tổng thời gian chạy thuật toán là $O(m^n \times n)$, trường hợp xấu nhất, mỗi đỉnh là một cộng đồng thì độ phức tạp là $O(m^3)$. Như vậy dường như rất khó khả thi khi chạy trên mạng có kích thước lớn và phức tạp.
2. Với phương pháp này thì toàn bộ cộng đồng thu được đều riêng biệt. Do vậy không thể làm việc hiệu quả trên mạng có các cộng đồng chồng chéo.

1.3.2 Phương pháp CONGA

Phương pháp CONGA viết tắt của từ Cluster Overlap Newman Algorithm là một phương pháp cải tiến của phương pháp Girvan-Newman nhằm khắc phục nhược điểm không phát hiện được các cộng đồng chồng chéo nhau của thuật toán cũ. Các đỉnh sẽ được nhân bản hoặc gộp lại dựa vào một độ đo nhằm tính số cộng đồng mà đỉnh đó thuộc. Gregory [2007] [7] đã đề xuất một độ đo đó được gọi là độ trung gian phân chia của đỉnh là số đường đi ngắn nhất mà chạy hai phần của đỉnh sau khi được phân chia. Như vậy với sự nâng cấp này phương pháp CONGA đã giải quyết được vấn đề chồng chéo cộng đồng, tuy nhiên vẫn tồn tại nhược điểm của phương pháp Girvan-Newman vẫn có độ phức tạp tính toán lớn.

1.3.3 Affiliation Graph Model (AGM)

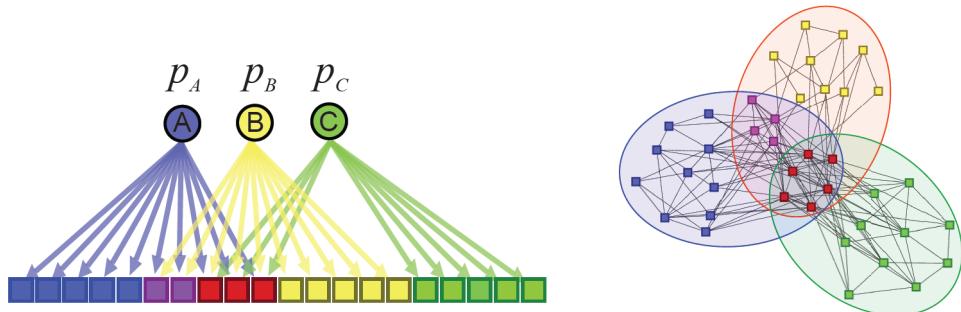
AGM được giới thiệu bởi Yang and Leskovec [2012] [15] là mô hình xác suất xây dựng dựa trên một đồ thị hai phần (bigraph) $B(\mathcal{V}, \mathcal{C}, M)$. Trong đó phần thứ nhất của đồ thị hai phần là các đỉnh \mathcal{V} của mạng và phần thứ 2 chính là các đỉnh định danh cho các cộng đồng trong mạng \mathcal{C} và với mỗi cộng đồng sẽ được gắn kèm với một xác suất p_A kết nối giữa cộng đồng với các đỉnh được thể hiện qua cạnh nối M giữa hai phần của đồ thị. AGM có khả năng phát hiện cộng đồng có tính chồng chéo không chồng chéo và lồng nhau trên mạng có kích thước lớn tương đối hiệu quả. Hình 1.6 sẽ minh họa cho mô hình.

Mục tiêu của phương pháp phát hiện cộng đồng trên mô hình AGM là tìm $\theta = B(\mathcal{V}, \mathcal{C}, M, \{p_c\})$:

$$\arg \max_{B(\mathcal{V}, \mathcal{C}, M, \{p_c\})} \prod_{(u,v) \in \mathcal{E}} p(u, v) \prod_{(u,v) \notin \mathcal{E}} (1 - p(u, v)) \quad (1.1)$$

Ở đó:

$$p(u, v) = 1 - \prod_{i \in C_{uv}} (1 - p_i)$$



Hình 1.6: Minh họa mô hình AGM

Tuy nhiên, theo tác giả để tìm được $B(\mathcal{V}, \mathcal{C}, M, \{p_c\})$ theo công thức 1.1 không hệ đơn giản và độ chính xác vẫn còn chưa cao.

1.3.4 Non-negative Matrix Factorization

Non-negative Matrix Factorization (NMF) hay còn gọi phân tích tích ma trận không âm thành nhân tử là một kỹ thuật được sử dụng phổ biến giải quyết các bài toán học máy (machine learning), học nhiều tầng (deep learning), hệ khuyến nghị (recommendation system),... Trong bài toán phát hiện cộng đồng, NMF tỏ ra khá vượt trội so với các phương pháp khác.

Trong khóa luận này, có đề cập đến mô hình BigCLAM của Yang and Leskovec [2013] [16] một sự nâng cấp từ mô hình AGM xây dựng dựa trên phương pháp NMF không chỉ trong bài phát hiện cộng đồng chồng chéo mà còn phát hiện được các cộng đồng không chồng chéo và

cộng đồng lồng nhau với kết quả vượt trội hơn rất nhiều so với AGM. Mô hình BigCLAM xây dựng một đồ thị hai phần (bipartite graph) giống như AGM phần thứ nhất của đồ thị là các nốt định danh cho các cộng đồng trong mạng phần còn lại là các đỉnh của mạng và cạnh nối giữa các hai phần của đồ thị chính là độ đo liên kết của các đỉnh trong mạng với các cộng đồng được biểu diễn thành vector. Chi tiết mô hình sẽ được trình bày chi tiết trong chương 3.

Chương 2

Cơ sở lý thuyết

Trong chương này, tôi xin trình bày chi tiết các khái niệm cơ bản, thuật ngữ và các ký hiệu được sử dụng trong khóa luận. Tiếp theo, tôi sẽ đề cập đến một số nghiên cứu liên quan đến bài toán phát hiện cộng đồng.

2.1 Vector và ma trận

Dầu tiên, tôi sẽ trình bày hai khái niệm vector và ma trận không âm.

Định nghĩa 1 (*Vector và ma trận không âm*)

Một vector $v = (v_i) \in \mathbb{R}^n$ là một vector không âm nếu tất cả các phần tử trong v là không âm. ($v_i \geq 0 \forall i \in \{1, \dots, n\}$)

Tương tự, một ma trận $F = (F_{ij}) \in \mathbb{R}^{n \times m}$ là không âm nếu mọi phần tử trong F là không âm. ($F_{ij} \geq 0 \forall i \in \{1, \dots, n\}, j \in \{1, \dots, m\}$)

2.2 Lý thuyết đồ thị

Trong mục này, tôi sẽ cung cấp một số định nghĩa về lý thuyết đồ thị được sử dụng trong chương 3 và 4.

Về cơ bản thì mạng tương tác thường được biểu diễn như một đồ thị $G(\mathcal{V}, \mathcal{E})$ gồm \mathcal{V} là tập các đỉnh, và \mathcal{E} là tập các cặp không có thứ tự gồm hai phần tử khác nhau của \mathcal{V} gọi là tập cạnh. Chúng ta thường sử dụng hai thuật ngữ nút (node), đỉnh (vertex) để ám chỉ cho \mathcal{V} và cạnh (edge), liên kết (link) để ám chỉ cho \mathcal{E} . Ngoài ra chúng còn được biểu diễn thông qua một ma trận kề (adjacency matrix) được phát biểu qua định nghĩa 2:

Định nghĩa 2 (*Ma trận kề cho một đồ thị vô hướng*)

Cho đồ thị $G = G(\mathcal{V}, \mathcal{E})$ là đồ thị vô hướng, một ma trận $A = (A_{ij}) \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ ($i, j \in \mathcal{V}$) được gọi là ma trận kề của G nếu:

$$A_{ij} = \begin{cases} 1 & \text{Nếu } (i, j) \in \mathcal{E} \\ 0 & \text{Ngược lại} \end{cases} \quad (2.1)$$

Trong lý thuyết đồ thị chúng ta sẽ sử dụng một số tính chất quan trọng để xây dựng mô hình phân chia đồ thị thành các đồ thị con ở đó số liên kết (cạnh) giữa chúng là nhỏ nhất và sự liên kết giữa các cá thể trong mỗi đồ thị con là lớn nhất. Việc phân chia đồ thị thành các đồ thị con như vậy vô cùng thuận lợi cho việc khởi tạo đầu vào cho phương pháp phát hiện cộng đồng được trình bày trong chương 3. Ta có định nghĩa 3 được đề xuất bởi Gleich and Seshadhri [2011] [6]

Định nghĩa 3 (Độ dẫn của một đồ thị con trong đồ thị vô hướng)

Cho đồ thị $G = G(\mathcal{V}, \mathcal{E})$ là đồ thị vô hướng, $A = (A_{ij})$ là một ma trận kề của G . $\mathcal{S} \subseteq \mathcal{V}$ là một tập đỉnh con trong đồ thị G , và $\bar{\mathcal{S}} = \mathcal{V} \setminus \mathcal{S}$ là tập đỉnh còn lại của đồ thi. Độ dẫn của \mathcal{S} là:

$$\varphi(\mathcal{S}) = \frac{\sum_{i \in \mathcal{S}, j \notin \mathcal{S}} A_{ij}}{\min(\mathcal{A}(\mathcal{S}), \mathcal{A}(\bar{\mathcal{S}}))} \quad (2.2)$$

Trong đó $\mathcal{A}(\mathcal{S}) = \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{V}} A_{ij}$

Ta có thể coi $\varphi(\mathcal{S})$ là độ dẫn của các cạnh từ các đỉnh trong \mathcal{S} đến các đỉnh bên ngoài \mathcal{S} .

Trong khóa luận này, tôi có sử dụng ba khái niệm là đỉnh lân cận, vùng lân cận và vùng lân cận cục bộ nhỏ nhất được trình bày trong định nghĩa 4

Định nghĩa 4 (Đỉnh lân cận, vùng lân cận và vùng lân cận cục bộ nhỏ nhất) Cho đồ thị $G = G(\mathcal{V}, \mathcal{E})$ ta có:

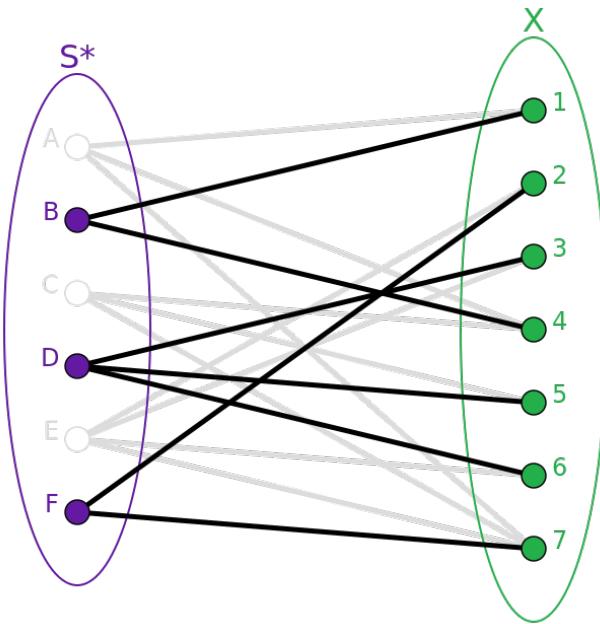
$\mathcal{N}(u) = \{v : v \in \mathcal{V}, (u, v) \in \mathcal{E}\}$ là tập các đỉnh lân cận với u .

$N(u) = \{u\} \cup \mathcal{N}(u)$ là vùng chứa các đỉnh lân cận của u .

Và một $N(u)$ được gọi là vùng lân cận cục bộ nhỏ nhất nếu nó có độ dẫn nhỏ hơn trong tất cả $N(v), v \in \mathcal{N}(u)$. Tức $\varphi(N(u)) \leq \varphi(N(v)), \forall v \in \mathcal{N}(u)$.

Dưới đây sẽ là một vài khái niệm trong lý thuyết đồ thị được nhắc đến trong khóa luận:

- **Đồ thị hai phần:** hay còn được gọi là đồ thị phân đôi. Đồ thị $G(\mathcal{V}, \mathcal{E})$ được gọi là một đồ thị phân đôi, nếu các đỉnh trong tập đỉnh \mathcal{V} có thể chia thành hai tập không giao nhau $\mathcal{V}_1, \mathcal{V}_2$ thỏa mãn không có cạnh nối giữa hai đỉnh bất kỳ thuộc cùng một tập. Được minh họa trong hình 2.1.



Hình 2.1: Đồ thị hai phần

- **Đồ thị có hướng và vô hướng:** Đồ thị $G(\mathcal{V}, \mathcal{E})$ là đồ thị vô hướng, nếu $\forall (u, v) \in \mathcal{E} \Leftrightarrow (v, u) \in \mathcal{E}$, các cạnh là các cặp cạnh không có thứ tự của các đỉnh. Nếu các cạnh là các cặp cạnh có thứ tự, ví dụ $(u, v) \in \mathcal{E}$ không nhất thiết $(v, u) \in \mathcal{E}$, thì đồ thị $G(\mathcal{V}, \mathcal{E})$ là đồ thị có hướng.
- **Mạng cá nhân - ego-network:** Một ego-network của đỉnh $u \in G(\mathcal{V}, \mathcal{E})$ là một đồ thị con dẫn xuất của $G(\mathcal{V}, \mathcal{E})$ bao gồm cả những thành phần lồng giềng của u .

2.3 Một số phương pháp ước lượng và tối ưu tham số

Trong mục này, tôi sẽ trình bày ngắn gọn về phương pháp ước lượng bằng cực đại khả dĩ và một số phương pháp tối ưu hàm lồi.

2.3.1 Ước lượng bằng cực đại khả dĩ

Ước lượng bằng cực đại khả dĩ (Tiếng anh thường được viết là MLE viết tắt của từ Maximum-Likelihood Estimation) là một kỹ thuật trong thống kê toán dùng để ước lượng tham số của một mô hình xác suất sử dụng phép lấy cực đại của hàm khả dĩ (likelihood function) của Fisher [3]. Định nghĩa 5 dưới đây sẽ mô tả rõ phương pháp.

Định nghĩa 5 (*Ước lượng bằng cực đại khả dĩ*) Giả sử $X = x_1, \dots, x_n$ là tập n quan sát và $Y = y_1, \dots, y_m$ là số nhãn của quan sát. x, y là hai biến độc lập ngẫu nhiên. Ta cần phải tìm

tham số θ để biểu thức sau đây đạt giá trị lớn nhất:

$$h_\theta(X) = P(Y|X; \theta) \quad (2.3)$$

nói cách khác:

$$\hat{\theta} = \arg \max_{\theta} P(Y|X; \theta) \quad (2.4)$$

Do các quan sát là biến độc lập ngẫu nhiên, ta có thể viết lại thành:

$$P(Y|X; \theta) = \prod_{i=1}^N P(y_i|x_i; \theta) \quad (2.5)$$

Nhưng trực tiếp tối ưu hàm số trên theo θ không hề đơn giản, hơn nữa khi N lớn thì tích của N số nhỏ hơn một có thể dẫn đến sai số trong tính toán. Một phương pháp thường được sử dụng đó là lấy logarit tự nhiên (cơ số e) của hàm khả dĩ ta được:

$$l(P(Y|X; \theta)) = \log \prod_{i=1}^N P(y_i|x_i; \theta) = \sum_{i=1}^N \log P(y_i|x_i; \theta) \quad (2.6)$$

2.3.2 Các phương pháp tối ưu hàm lồi

Trong bài toán tối ưu, chúng ta đặc biệt quan tâm tới những bài toán mà hàm mục tiêu là một hàm lồi. Định nghĩa 6 sẽ cho thấy khái niệm về tập lồi và hàm lồi và hàm lõm [5]

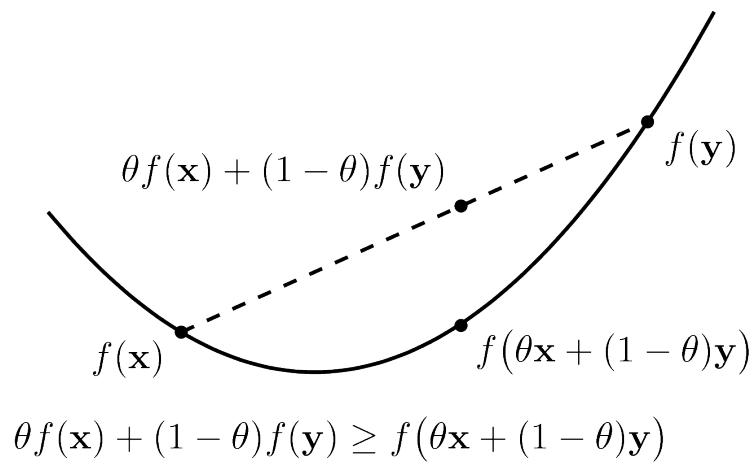
Định nghĩa 6 (Tập lồi và hàm lồi)

Một tập hợp C được gọi là tập lồi nếu với hai điểm bất kỳ $x_1, x_2 \in C$, điểm $x_\theta = \theta x_1 + (1-\theta)x_2$ cũng nằm trong C với bất kỳ $0 \leq \theta \leq 1$.

Và một hàm số $f : \mathbb{R}^n \rightarrow \mathbb{R}$ được gọi là hàm lồi (convex function) nếu $\text{dom } f$ là một tập lồi và: (hình 2.2)

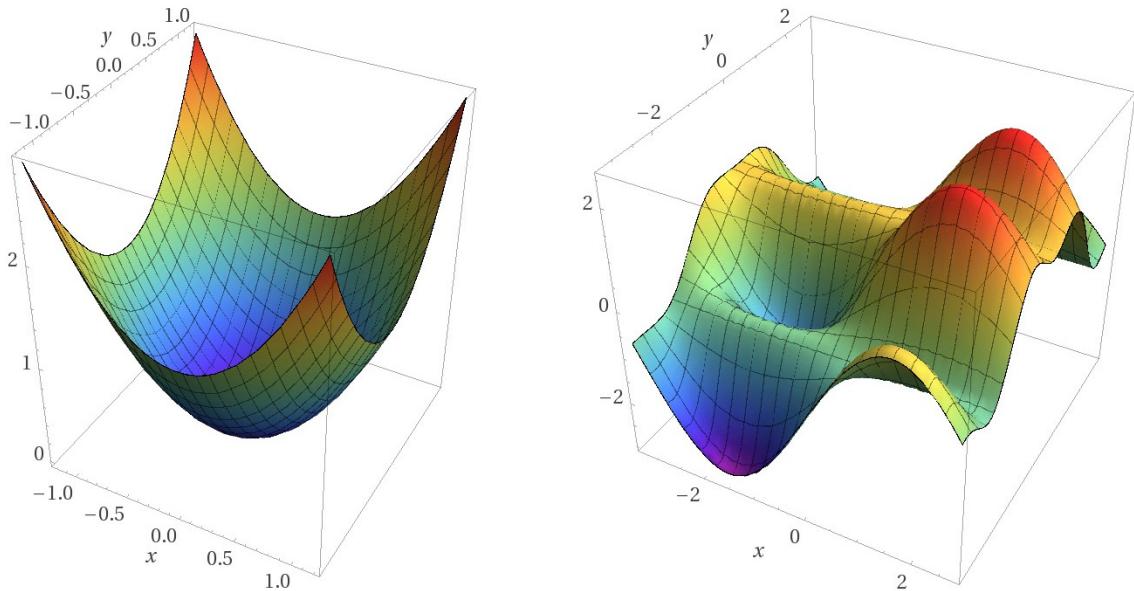
$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y); \forall x, y \in \text{dom } f, 0 \leq \theta \leq 1 \quad (2.7)$$

Điều kiện $\text{dom } f$ là một tập lồi là rất quan trọng để định nghĩa được $f(\theta x + (1 - \theta)y)$.



Hình 2.2: Hàm lồi.

Trong khóa luận này có đề cập đến tối ưu cực đại hàm lõm. Do vậy nếu $-f(x)$ là hàm lồi thì $f(x)$ là hàm lõm. Với lợi thế của bài toán có hàm mục tiêu là hàm lồi/ lõm ta luôn luôn tìm được điểm cực đại hoặc cực tiểu toàn cục (global minimum/ maximum). Hình 2.3 minh họa hàm lồi và không lồi. Dễ dàng thấy được ở hàm lồi luôn chỉ có duy nhất một điểm cực tiểu ngược lại ở hàm không lồi luôn có các điểm cực tiểu cục bộ và cực đại cục bộ như vậy rất khó khăn hoặc bất khả thi trong việc xác định điểm cực tiểu tốt nhất cho bài toán.



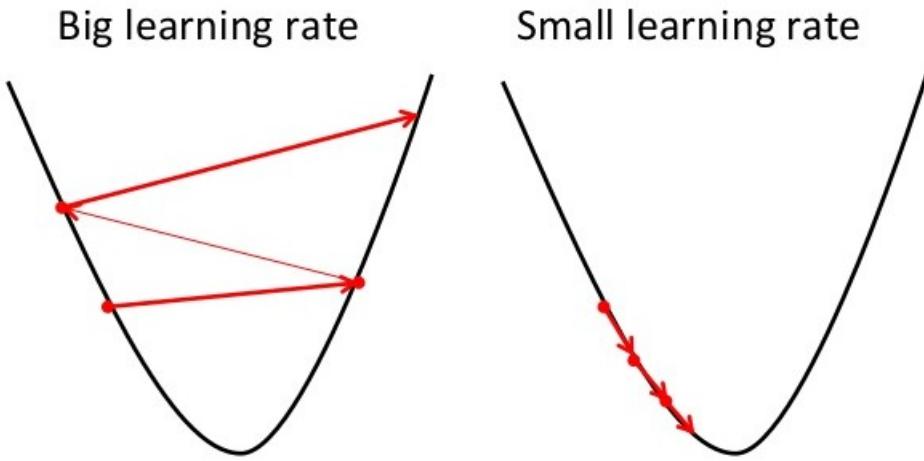
Hình 2.3: Hàm lồi (trái) và hàm không lồi (phải)

Một trong những phương pháp tối ưu hàm lồi được sử dụng rộng rãi cho học máy chính là phương pháp tăng giảm gradient nhằm ước lượng tham số θ của mô hình, khi $J(\theta)$ khả vi. Phương pháp này được trình bày trong thuật toán 1 [5] dưới đây:

Algorithm 1 Batch gradient descent method.

- 1: Given bắt đầu từ một điểm $\theta = \{\theta_1, \dots, \theta_n\} \in \text{dom } f$
 - 2: repeat
 - 3: 1. $\Delta\theta := -\nabla_\theta f(\theta)$.
 - 4: 2. Chọn bước nhảy α mặc định hoặc thông qua thuật toán backtracking line search
 - 5: 3. Cập nhật. $\theta := \theta + \alpha\Delta\theta$
 - 6: until thuật toán hội tụ.
-

Trong thuật toán 1 có sử dụng phương pháp xác định α là tham số thể hiện bước nhảy hay tốc độ huấn luyện của thuật toán. Hình 2.4 khi α quá lớn hoặc quá nhỏ đều sẽ làm ảnh hưởng đến thời gian hội tụ của bài toán. Vì vậy trong các thuật toán giảm gradient, tốc độ học α cần được thay đổi phù hợp theo mỗi lần lặp. Thuật toán 2 [11] được sử dụng phổ biến để xác định giá trị α qua mỗi lần lặp, có dễ hiểu hơn về phương pháp thông qua cách nghĩ trước nước đi trong chơi cờ.



Hình 2.4: Sự ảnh hưởng của α đến quá trình huấn luyện.

Algorithm 2 Backtracking line search.

Given bắt đầu tại một điểm $\Delta\theta$ với f tại $\theta \in \text{dom } f, \gamma \in (0, 0.5), \beta \in (0, 1)$
While $f(\theta + \alpha \Delta\theta) > f(\theta) + \gamma\alpha\nabla f(\theta)^T \Delta\theta$
 $\alpha = \beta\alpha$

Với hàm mục tiêu có độ phức tạp thì việc tính $\nabla f(x)$ đòi hỏi khối lượng tính toán lớn. Và nếu dữ liệu có kích thước N lớn thì mỗi bước lặp có khối lượng tính toán lớn, thuật toán chạy chậm. Do vậy để tiết kiệm khối lượng tính toán, thuật toán 3 [4] phương pháp giảm gradient ngẫu nhiên là một phương pháp phù hợp trong huấn luyện dữ liệu lớn.

Algorithm 3 Stochastic Gradient descent method.

```
1: Given bắt đầu tại  $\theta = \{\theta_1, \dots, \theta_n\} \in \text{dom } f$ 
2: repeat
3:   1. Xáo trộn tập dữ liệu huấn luyện
4:   for  $i = 1 : N$  do
5:     1.  $\Delta\theta := -\nabla_{\theta}f(\theta; x^i, y^i)$ .
6:     2. Chọn bước nhảy  $\alpha$  mặc định hoặc thông qua thuật toán backtracking line search
7:     3. Cập nhật.  $\theta := \theta + \alpha\Delta\theta$ 
8:   end for
9: until thuật toán hội tụ.
```

Thuật toán 4 [10] là sự kết hợp của hai thuật toán trên phù hợp chạy trên những hệ thống tính toán song song/ phần tán được đề cập trong chương 4.

Algorithm 4 Mini-batch Stochastic gradient descent method.

```
1: Given bắt đầu tại  $\theta = \{\theta_1, \dots, \theta_n\} \in \text{dom } f$  và chọn  $1 \leq k \leq N$ 
2: repeat
3:   1. Xáo trộn tập dữ liệu huấn luyện
4:   for  $i = 1; i \leq N; i+ = k$  do
5:     1.  $\Delta\theta := -\nabla_{\theta}f(\theta; x^{i:i+k}, y^{i:i+k})$ .
6:     2. Chọn bước nhảy  $\alpha$  mặc định hoặc thông qua thuật toán backtracking line search
7:     3. Cập nhật.  $\theta := \theta + \alpha\Delta\theta$ 
8:   end for
9: until thuật toán hội tụ.
```

Chương 3

Phương pháp phát hiện cộng đồng dựa trên mô hình BigCLAM

Một trong những vấn đề của các thuật toán phát hiện cộng đồng hiện nay là khả năng xử lý các mạng lớn lên tới hàng triệu đỉnh và cạnh nối. Đặc biệt là bài toán phát hiện cộng đồng chồng chéo càng là một vấn đề. Năm 2012, đồng tác giả hai nhà khoa học Yang and Leskovec [2012] [15] đã đề giới thiệu mô hình AGM (Community-Affiliation Graph Model) cho phương pháp phát hiện cộng đồng chồng chéo. Và một năm sau đó họ đã đề xuất phương pháp phát hiện cộng đồng lớn dựa trên mô hình AGM và đặt tên là mô hình BigCLAM (Community Affiliation Model for BigNetworks).

Trong chương này, tôi sẽ trình bày chi tiết phương pháp phát hiện cộng đồng có tính chồng chéo dựa trên mô hình BigCLAM được đề suất bởi Yang and Leskovec [2013] [16].

3.1 Mô hình BigCLAM

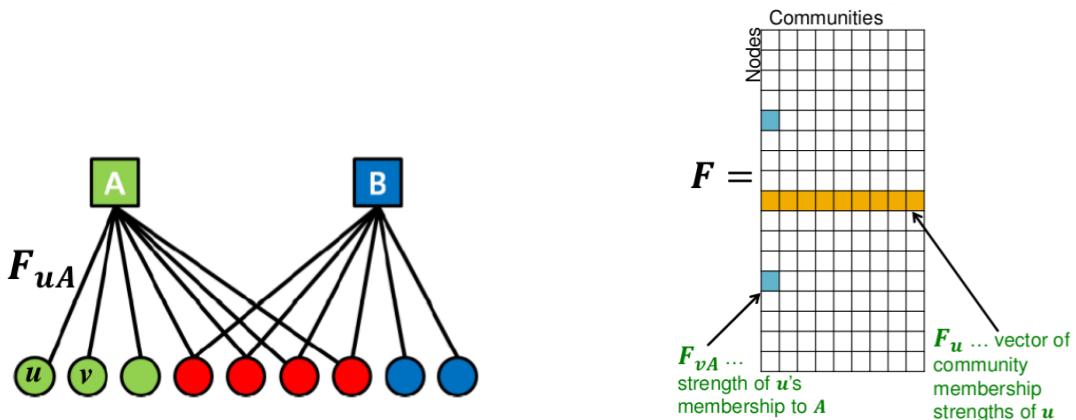
Trong phần này, tôi sẽ trình bày ngắn gọn mô hình cộng đồng cho đồ thị/ mạng mà BigCLAM sử dụng. Mô hình BigCLAM phần lớn vẫn dựa trên mô hình hai nhà khoa học đề xuất là AGM, một đồ thị hai phần thể hiện sự liên kết giữa các cộng đồng và các đỉnh trong mạng (hình 3.1), được ký hiệu là $B(\mathcal{V}, \mathcal{C}, \mathcal{M})$. Mặc dù không giống như AGM, BigCLAM không có xác suất kết nối trên mỗi cộng đồng, nhưng thay vào đó từ mỗi cộng đồng sẽ có một trọng số liên kết đến từng phần tử. Mô hình gán một trọng số không âm $F_{vc}(v \in \mathcal{V}, c \in \mathcal{C})$ trên mỗi cạnh của đồ thị phân đôi được thể hiện như là độ liên kết của quan hệ giữa một đỉnh trong mạng đến cộng đồng.

Định nghĩa 7 (*Ma trận và vector trọng số kết nối cộng đồng*)

Một matrix không âm $F = (F_{vc}) \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{K}|}$ ($F_{vc} \geq 0$) biểu diễn các trọng số liên kết giữa các đỉnh đến các cộng đồng. Một vector không âm $F_u = (F_{uc} \geq 0) \in \mathbb{R}^{|\mathcal{V}|}$ biểu diễn các trọng

số kết nối giữa đỉnh u đến tất cả các cộng đồng (Hình 3.1). Với F , mô hình BigCLAM sinh đồ thị $G(\mathcal{V}, \mathcal{E})$ bằng tạo các cạnh (u, v) với xác suất $p(u, v)$:

$$p(u, v) = 1 - \exp(-F_u \cdot F_v^T) = 1 - \exp\left(-\sum_{c \in \mathcal{C}} F_{uc} F_{vc}\right) \quad (3.1)$$



Hình 3.1: Mô hình cộng đồng BigCLAM Hình 3.2: Ma trận trọng số kết nối cộng đồng

Hay nói cách khác, công thức 3.1 được hiểu như sau. Ta giả sử rằng mỗi cộng đồng $c \in \mathcal{C}$ kết nối đến các thành viên của nó $u, v \in \mathcal{V}$ hoàn toàn độc lập với xác suất là $1 - \exp(-F_u \cdot F_v)$ thì xác suất tạo ra cạnh $(u, v) \in \mathcal{E}$ sẽ là $1 - \exp(-F_{uc} \cdot F_{vc})$. Định nghĩa này dựa trên quan sát của tác giả cho rằng với mỗi tập đỉnh là thành phần chung của sự chồng chéo các cộng đồng sẽ thu được xác suất cao hơn các tập không chồng chéo.

Công thức 3.1 rõ ràng cho thấy BigCLAM tạo nên sự chồng chéo của các cộng đồng. Nó tạo ra một mối quan hệ giữa xác suất cạnh và số lượng cộng đồng chia sẻ. Điều này là do thực tế là các đỉnh chia sẻ nhiều thành viên cộng đồng nên nhận được nhiều cơ hội để tạo một cạnh trong đồ thị. Ví dụ, các cặp nút màu đỏ trong vùng chồng chéo của các cộng đồng A và B trong hình 3.1 có hai cơ hội để tạo ra một cạnh. Đầu tiên chúng tạo ra một cạnh với xác suất $1 - \exp(-F_{uA} \cdot F_{vA})$ (do là thành viên của cộng đồng A) và cũng tạo ra một cạnh với xác suất $1 - \exp(-F_{uB} \cdot F_{vB})$ (do là thành viên của cộng đồng B). Như vậy xác suất cạnh giữa các đỉnh này là $1 - \exp(-F_{uA} \cdot F_{vA} + F_{uB} \cdot F_{vB})$.

3.2 Phát hiện cộng đồng sử dụng mô hình BigCLAM

Với đồ thị $G(\mathcal{V}, \mathcal{E})$ cho trước, mục tiêu là phát hiện ra K cộng đồng bằng cách tìm matrix $\hat{F} \in \mathbb{R}^{|\mathcal{V}| \times K}$. Trong mục 3.2.1 sẽ trình bày phương pháp xác định \hat{F} thông qua việc tìm cực đại hàm khả dĩ (định nghĩa 5) $l(F) = \log P(G, F)$

3.2.1 Cực đại hàm khả dĩ ma trận trọng số liên kết của đồ thị

Cho $G(\mathcal{V}, \mathcal{E})$ là một đồ thị với tập cạnh \mathcal{E} và tập đỉnh \mathcal{V} , với $F = (F_{vc}) \in \mathbb{R}^{|\mathcal{V}| \times |K|}$ ($v \in \mathcal{V}, c \in \mathcal{C}$) là một matrix trọng số liên kết (ở đây K là số cộng đồng).

Xét toàn bộ đồ thị $G(\mathcal{V}, \mathcal{E})$ ta có được $P(G|F)$ chính là khả dĩ của F sinh ra đồ thị G như sau:

$$\begin{aligned}\mathcal{L}(F|G) = P(G|F) &= \begin{cases} p(u, v) & \text{Nếu } (u, v) \in \mathcal{E} \\ 1 - p(u, v) & \text{Ngược lại} \end{cases} \\ &= \prod_{(u,v) \in \mathcal{E}} p(u, v) \prod_{(u,v) \notin \mathcal{E}} (1 - p(u, v)) \\ &= \prod_{(u,v) \in E} (1 - \exp(-F_u F_v^T)) \prod_{(u,v) \notin \mathcal{E}} \exp(-F_u F_v^T).\end{aligned}\tag{3.2}$$

Đây là một dạng bài toán ước lượng tham số bằng cực đại hàm khả dĩ (Maximum likelihood estimation). Theo định nghĩa 5 ta được hàm log-likelihood $l(F)$ sau:

$$l(F) = \log \mathcal{L}(F|G) = \sum_{(u,v) \in \mathcal{E}} (\log(1 - \exp(-F_u F_v^T))) - \sum_{(u,v) \notin \mathcal{E}} \exp(F_u F_v^T).\tag{3.3}$$

Mặt khác hàm logarit là đơn điệu tăng nên \hat{F} cực đại $l(F)$ thì cũng là cực đại log $P(G|F)$ tức là:

$$\hat{F} = \arg \max_{F \geq 0} \mathcal{L}(F) = \arg \max_{F \geq 0} l(F)\tag{3.4}$$

Từ công thức 3.3, ta giả sử với K cộng đồng đã biết (phương pháp tìm K sẽ được đề cập ở phần sau) ta cần giải quyết bài toán trong công thức 3.4. Nhưng thay vì phải cực đại $l(F)$ ta sẽ tách thành các bài toán tìm cực đại $l(F_u)$ và cập nhật lại F_u trên từng đỉnh $u \in \mathcal{V}$ theo các $F_v, v \neq u$ như sau:

$$\hat{F}_u = \arg \max_{F_{uc}, c \in \mathcal{C}} l(F_u)\tag{3.5}$$

Trong đó:

$$l(F_u) = \sum_{v \in \mathcal{N}(u)} \log(1 - \exp(-F_u F_v^T)) - \sum_{v \notin \mathcal{N}(u)} F_u F_v^T\tag{3.6}$$

$\mathcal{N}(u)$ là tập các đỉnh lân cận của u (Định nghĩa 4)

Ta thấy rằng $l(F_u)$ là một hàm lõm như vậy vẫn đề trên được coi như là bài toán tối ưu hàm lõm và có thể tìm được điểm cực đại duy nhất một cách dễ dàng. Để ước lượng được các vector F_u (công thức 3.5) ta sử dụng các phương pháp tối ưu hàm lồi/ lõm được trình bày ở mục 2.3.2

$$l'(F_u) = l(F_u) + \alpha \bigtriangledown (l(F_u))\tag{3.7}$$

Trong đó gradient cho log-likelihood của F_u là:

$$\nabla(l(F_u)) = \frac{\partial l(F_u)}{\partial F_u} = \sum_{v \in \mathcal{N}(u)} F_v \frac{\exp(-F_u F_v^T)}{1 - \exp(-F_u F_v^T)} - \sum_{v \notin \mathcal{N}(u)} F_v \quad (3.8)$$

Để giảm độ phức tạp tính toán xuống còn $O(|\mathcal{N}(u)|)$ tôi xin biến đổi lần lượt các công thức 3.6 3.8 như sau:

$$\sum_{v \notin \mathcal{N}(u)} F_u F_v^T = \sum_v F_u F_v^T - F_u F_u^T - \sum_{v \in \mathcal{N}(u)} F_u F_v^T \quad (3.9)$$

$$\sum_{v \notin \mathcal{N}(u)} F_v = \left(\sum_v F_v \right) - F_u - \left(\sum_{v \in \mathcal{N}(u)} F_v \right) \quad (3.10)$$

Ma trận F được cập nhật sau mỗi vòng lặp theo công thức 3.7. Ta nên cho thuật toán hội tụ khi:

$$\frac{l'(F_u) - l(F_u)}{l(F_u)} < 0.0001, \forall u \in \mathcal{V} \quad (3.11)$$

3.2.2 Khởi tạo ma trận trọng số liên kết cộng đồng

Kết quả của bài toán phát hiện cộng đồng ảnh hưởng rất nhiều từ việc khởi tạo giá trị $F_u, u \in \mathcal{V}$ cho quá trình tối ưu hàm lồi/ lõm. Để khởi tạo ma trận F (Định nghĩa 8), ta chọn sử dụng phương pháp tìm vùng lân cận cục bộ nhỏ nhất của Gleich and Seshadhri [2011] [6] đã được đề cập đến trong định nghĩa 3 và định nghĩa 4

Định nghĩa 8 (*Khởi tạo ma trận trọng số liên kết*)

Gọi F là một ma trận trọng số liên kết. F được khởi tạo như sau:

$$F_{(u')(N(u))} = \begin{cases} 1 & \text{Nếu } u' \in N(u) \text{ và } N(u) \text{ là vùng lân cận cục bộ nhỏ nhất của } u \\ 0 & \text{Trường hợp còn lại} \end{cases} \quad (3.12)$$

3.2.3 Xác định các liên kết cộng đồng

Sau khi công thức 3.4 hội tụ, ta sẽ có được một ma trận trọng số liên kết F . Tuy nhiên ta cần phải xác định đỉnh u có thuộc cộng đồng c hay không. Vì vậy ta phải lựa chọn một ngưỡng δ cho ma trận F qua định nghĩa 9.

Định nghĩa 9 (*Xác định liên kết cộng đồng*) Cho $G(\mathcal{V}, \mathcal{E})$, F là ma trận trọng số liên kết của G và $\epsilon = \frac{2|E|}{|V|(|V| - 1)}$.

Thuật toán xét $u \in \mathcal{V}$ như một thành viên trong cộng đồng $c \in \mathcal{C}$ nếu thỏa mãn $F_{uc} \geq \delta = \sqrt{-\log(1 - \epsilon)}$.

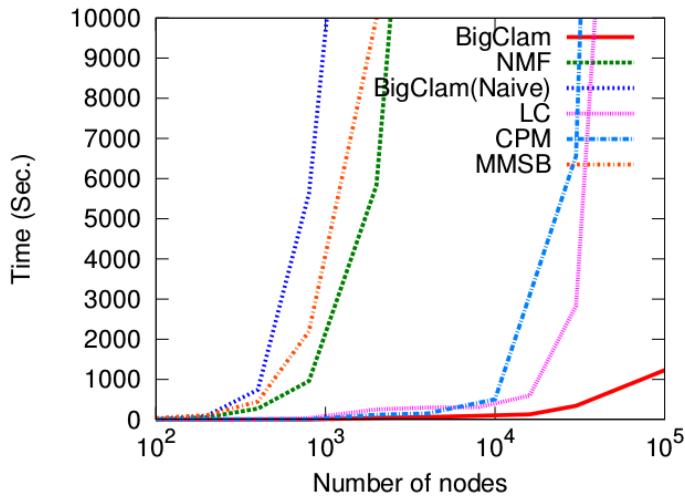
3.2.4 Xác định số cộng đồng

Trong phần đầu của chương này đã tiến hành xây dựng mô hình với giả sử K số cộng đồng trong $G(\mathcal{V}, \mathcal{E})$ đã biết. Yang và Leskovec đã trình bày một phương pháp để ước lượng gần nhất số cộng đồng K vẫn dựa trên phương pháp cực đại hàm khả dĩ như sau:

- Chọn ngẫu nhiên 20% cặp đỉnh của đồ thị G cho H ;
- Với mỗi K là số cộng đồng. $K \in [minCom; maxCom]$ với bước nhảy trong khoảng $[minCom; maxCom]$ là $\alpha = \exp\left(\log\left(\frac{maxCom}{minCom}\right) \times \frac{1}{divCom}\right)$. Trong đó $minCom$ là số cộng đồng nhỏ nhất, $maxCom$ là số cộng đồng lớn nhất và $divCom$ là số lượng K muốn thử.
 - Chạy thuật toán phát hiện cộng đồng, với K là số cộng đồng mong muốn phái hiện
 - Dánh giá likelihood của ma trận trọng số trên tập H , $l_k(F_H)$.
- $\hat{K} = \arg \max_K l_K(F_H)$.

3.3 Đánh giá hiệu năng

Theo kết quả của các tác giả khi so sánh hiệu năng giữa BigCLAM và các mô hình được sử dụng để phát hiện cộng đồng như LC [1], CPM [14], MMSB [2]. Hình 3.3 cho thấy BigCLAM cho hiệu năng cao hơn từ 10 đến 100 lần so với phương pháp còn lại.



Hình 3.3: So sánh hiệu năng của các mô hình với BigCLAM

Ngoài ra các tác giả sử dụng một số mạng đã được xác định trước các cộng đồng (ground-truth communities) để đánh giá độ chính xác của các mô hình được dùng để phát hiện cộng

đồng. Bảng 3.1 dưới đây là thông tin các mạng được sử dụng.

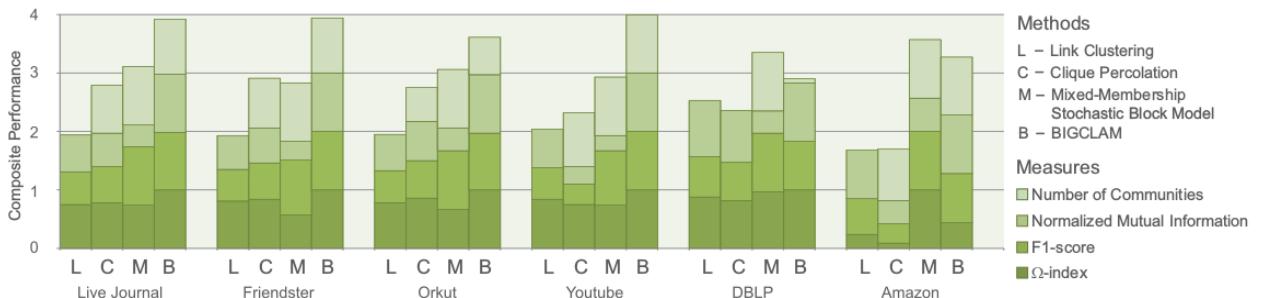
Bảng 3.1: Mô tả các mạng sử dụng để thực nghiệm

| Tên mạng | Số đỉnh | Số cạnh | Số cộng đồng |
|-------------------------|------------|---------------|--------------|
| Mạng xã hội LiveJournal | 3,997,962 | 34,681,189 | 287,512 |
| Mạng xã hội Friendster | 65,608,366 | 1,806,067,135 | 957,154 |
| Mạng xã hội Ourkut | 3,072,441 | 117,185,083 | 6,288,363 |
| Mạng xã hội Youtube | 1,134,890 | 2,987,624 | 8,385 |
| Mạng cộng tác DBLP | 317,080 | 1,049,866 | 13,477 |
| Mạng bán hàng Amazon | 334,863 | 925,872 | 75,149 |

Tác giả sử dụng các thang đo đánh giá như sau:

- **Average F1 score [8]** : Thể hiện mức độ chính xác trung bình của các cộng đồng phát hiện được với các cộng đồng thực tế.
- **Omega Index [9]**: Là chỉ số thể hiện độ chính xác số cộng đồng mà mỗi đỉnh thuộc.
- **Accuracy in the number of communities**: Độ chính xác về số lượng cộng đồng

Hình 3.4 là kết quả của sự so sánh hiệu năng giữa các mô hình trên một số mạng. Dễ dàng thấy được, hiệu năng trung bình của mô hình BigCLAM là 3.60 cao hơn hẳn các mô hình còn lại như Link Clustering (2.01), CPM (2.47) và MMSB (3.14). Nhìn chung, BigCLAM có hiệu năng tương đối tốt, tỏ ra khá vượt trội so với các phương pháp thường được sử dụng và nó có khả năng phát hiện đa dạng các loại cộng đồng (chồng chéo, không chồng chéo, lồng nhau).



Hình 3.4: Biểu đồ so sánh độ chính xác các mô hình trên các mạng biết trước cộng đồng

Mô hình BigCLAM dựa trên kỹ thuật NMF, vậy nên hoàn toàn có thể tăng tốc độ trên các kỹ thuật xử lý song song và phân tán. Trong chương 4 giới thiệu và đề xuất phương pháp tăng tốc bài toán dựa trên sức mạnh xử lý trên cụm máy tính.

Chương 4

Cài đặt phương pháp phát hiện cộng đồng sử dụng mô hình BigCLAM trên hệ thống phân tán

Chương này sẽ trình bày ngắn gọn về hệ thống tính toán và lưu trữ phân tán sử dụng Apache Spark/ Hadoop framework trong xử lý dữ liệu lớn. Cuối cùng, tôi sẽ trình bày quá trình cài đặt phương pháp phát hiện cộng đồng dựa trên mô hình BigCLAM trên Spark/ Hadoop.

4.1 Tổng quan hệ tính toán phân tán

4.1.1 Giới thiệu Apache Hadoop & Spark

Apache Hadoop¹, công nghệ được viết bởi Doug Cutting dựa trên hai bài báo GFS (Google File System) và MapReduce của Google vào năm 2005. Tháng Tư năm 2008, Hadoop trở thành hệ thống nhanh nhất để sắp xếp (sort) 1 terabyte dữ liệu, khi mất 209 giây chạy trên cluster gồm 910 nodes, đánh bại kỷ lục cũ là 297 giây. Tháng 11 năm 2008, Google thông báo hệ thống MapReduce của họ chỉ cần 68 giây để sắp xếp 1 terabyte dữ liệu. Đến tháng 5 năm 2009, Yahoo sử dụng Hadoop chỉ cần 62 giây để làm việc tương tự. Từ đó đến nay, cả một hệ sinh thái đã được xây dựng lấy Hadoop làm nòng cốt để giải quyết những bài toán về dữ liệu lớn.

Hadoop bao gồm hai thành phần chính (Hình 4.1):

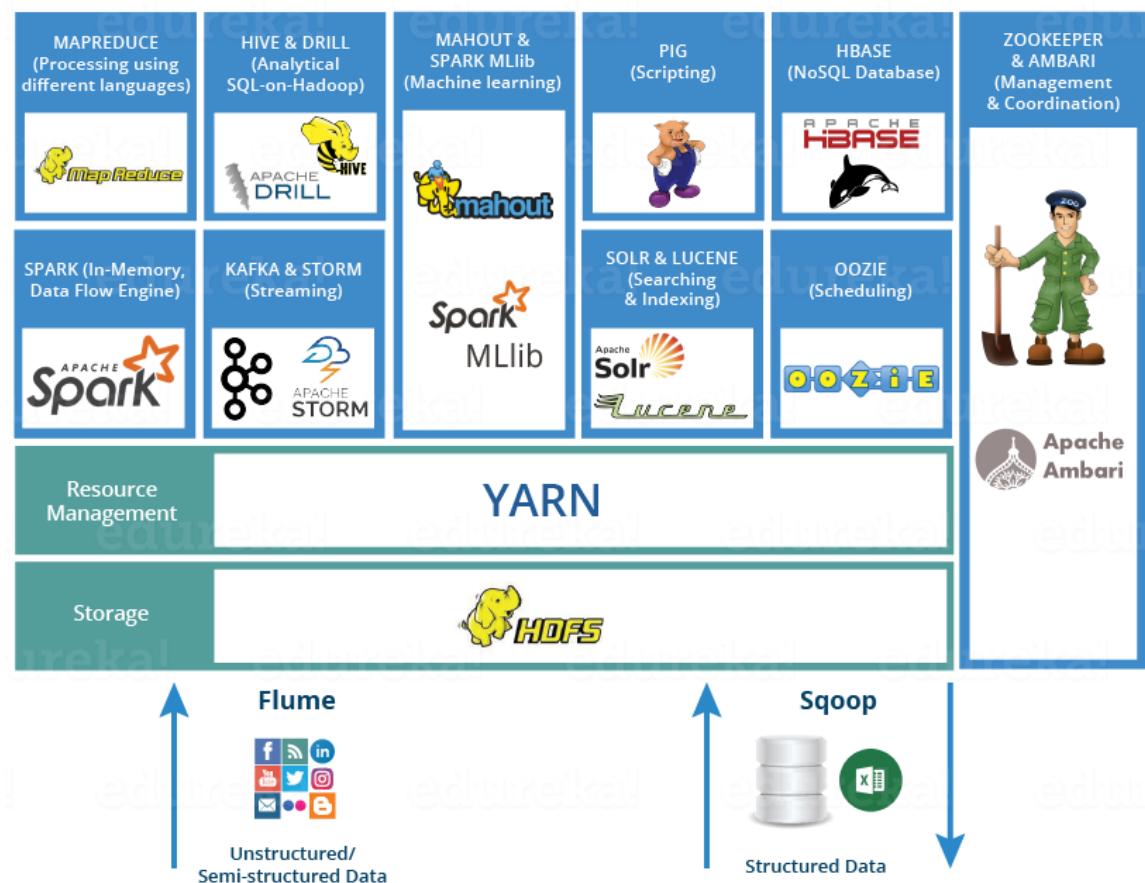
- **HDFS:** Hadoop Distributed File System, hệ thống tập tin phân tán, cho phép lưu trữ dữ liệu trên cluster gồm nhiều máy tính có cấu hình ở mức thông thường

¹<http://hadoop.apache.org/>

- **MapReduce framework:** cho phép xử lý dữ liệu song song trên cluster.

Trên nền tảng hai thành phần đó, cộng đồng mã nguồn mở đã phát triển thêm rất nhiều công cụ khác giúp tăng hiệu quả khi làm việc với Hadoop (Hình 4.1):

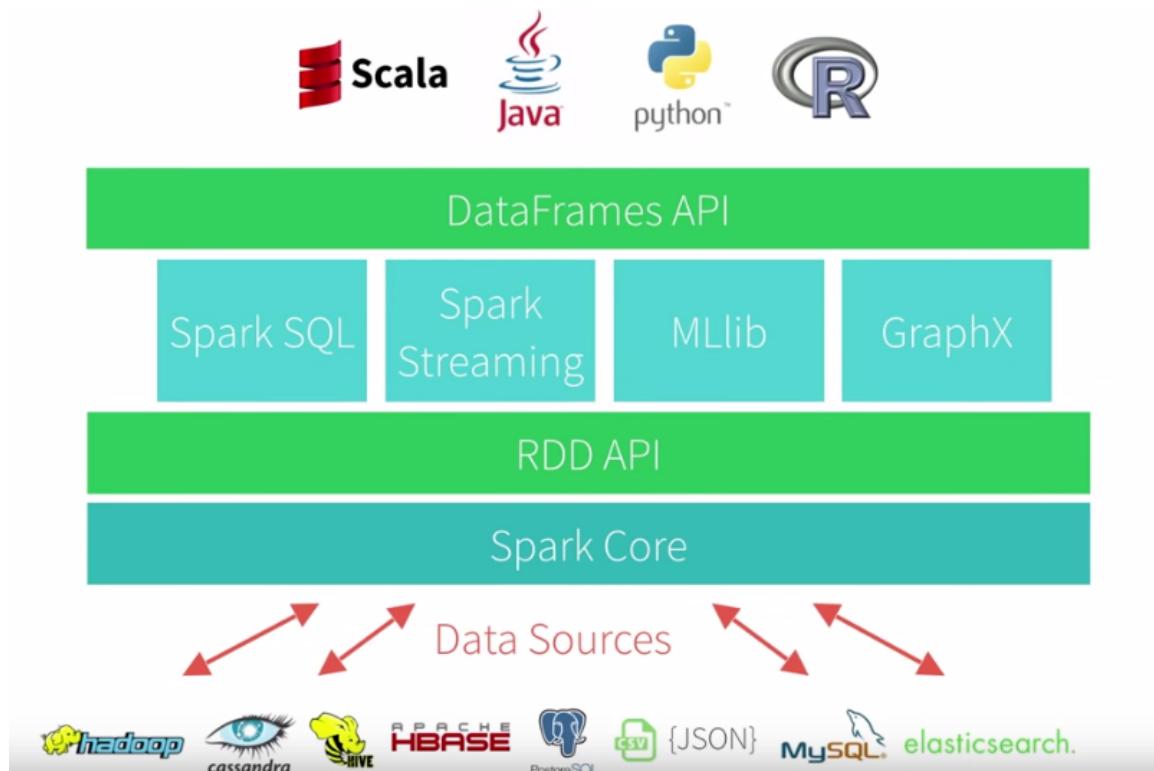
- Hbase: Cơ sở dữ liệu NoSQL, được xây dựng trên nền của HDFS, hỗ trợ những dữ liệu phi cấu trúc.
- Flume: Dùng để thu thập dữ liệu từ các nguồn như log hệ thống
- Oozie: Định nghĩa mối tương quan giữa các tác vụ và lập luồng làm việc cho MapReduce.
- Hive: Sử dụng những câu lệnh như SQL và biên dịch những câu lệnh này thành tập hợp các tác vụ MapReduce.
- Mahout: Sử dụng cho các bài toán về học máy.
- Sqoop: Dùng để chuyển đổi dữ liệu từ các cơ sở dữ liệu quan hệ sang HDFS.



Hình 4.1: Kiến trúc và hệ sinh thái của Apache Hadoop

Matei Zaharia, cha đẻ của Spark, sử dụng Hadoop từ những ngày đầu. Đến năm 2009 ông viết Apache Spark¹ để giải quyết những bài toán học máy ở đại học UC Berkely vì Hadoop MapReduce hoạt động không hiệu quả cho những bài toán này. Rất sớm sau đó ông nhận ra rằng Spark không chỉ hữu ích cho học máy mà còn cho cả việc xử lý luồng dữ liệu hoàn chỉnh. Spark cho phép thực hiện tính toán cùng lúc trên toàn bộ bộ tập dữ liệu mà không cần phải trích xuất mẫu tính toán thử nghiệm. Tốc độ xử lý của Spark có được do việc tính toán được thực hiện cùng lúc trên nhiều máy khác nhau. Đồng thời việc tính toán được thực hiện ở bộ nhớ trong (in-memories) hay thực hiện hoàn toàn trên RAM.

Khi ta có một tác vụ nào đó quá lớn mà không thể xử lý trên một máy tính/ server, Spark cho phép ta phân chia tác vụ này thành những phần dễ quản lý hơn. Sau đó, Spark sẽ chạy các tác vụ này trong bộ nhớ trên các cluster của nhiều máy tính/server khác nhau để khai thác tốc độ truy xuất nhanh từ RAM. Spark sử dụng API Resilient Distributed Dataset (RDD) để xử lý dữ liệu. Spark nhanh hơn khá nhiều so với cách tiếp cận MapReduce truyền thống. Theo như giới thiệu từ trang chủ của Apache Spark, thì tốc độ của nó nhanh hơn 100x so với Hadoop MapReduce khi chạy trên bộ nhớ, và nhanh hơn 10x lần khi chạy trên đĩa, tương thích hầu hết các CSDL phân tán (HDFS, HBase, Cassandra, ...). Ta có thể sử dụng Java, Scala, Python hoặc R để triển khai các ứng dụng trên Spark.



Hình 4.2: Kiến trúc và hệ sinh thái Apache Spark

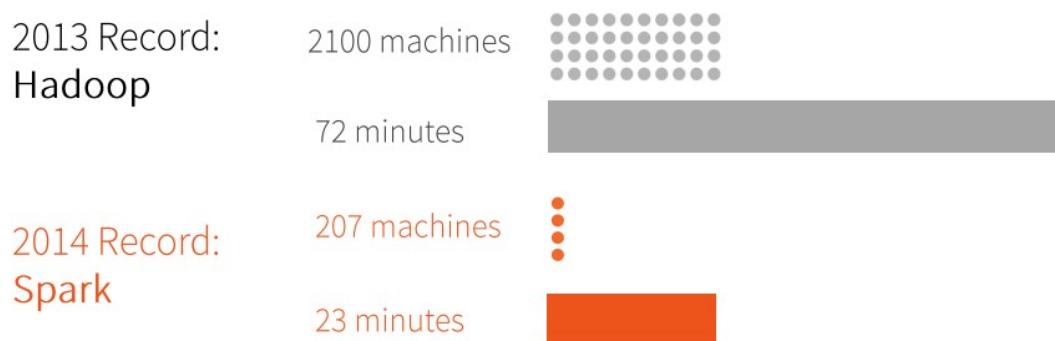
¹<http://spark.apache.org/>

Kiến trúc và hệ sinh thái của Apache gồm:

- **Spark Core**: cung cấp những chức năng cơ bản nhất của Spark như lập lịch cho các tác vụ, quản lý bộ nhớ, fault recovery, tương tác với các hệ thống lưu trữ... Đặc biệt, Spark Core cung cấp API để định nghĩa RDD (Resilient Distributed DataSet) là tập hợp của các item được phân tán trên các node của cluster và có thể được xử lý song song.
- **Spark SQL** cho phép truy vấn dữ liệu cấu trúc qua các câu lệnh SQL. Spark SQL có thể thao tác với nhiều nguồn dữ liệu như Hive tables, Parquet, và JSON;
- **Spark Streaming** cung cấp API để dễ dàng xử lý dữ liệu stream;
- **MLlib** Cung cấp rất nhiều thuật toán của học máy như: classification, regression, clustering, collaborative filtering...
- **GraphX** là thư viện để xử lý đồ thị.

Spark có thể chạy trên nhiều loại Cluster Managers như Hadoop YARN, Apache Mesos hoặc trên chính cluster manager được cung cấp bởi Spark được gọi là Standalone Scheduler.

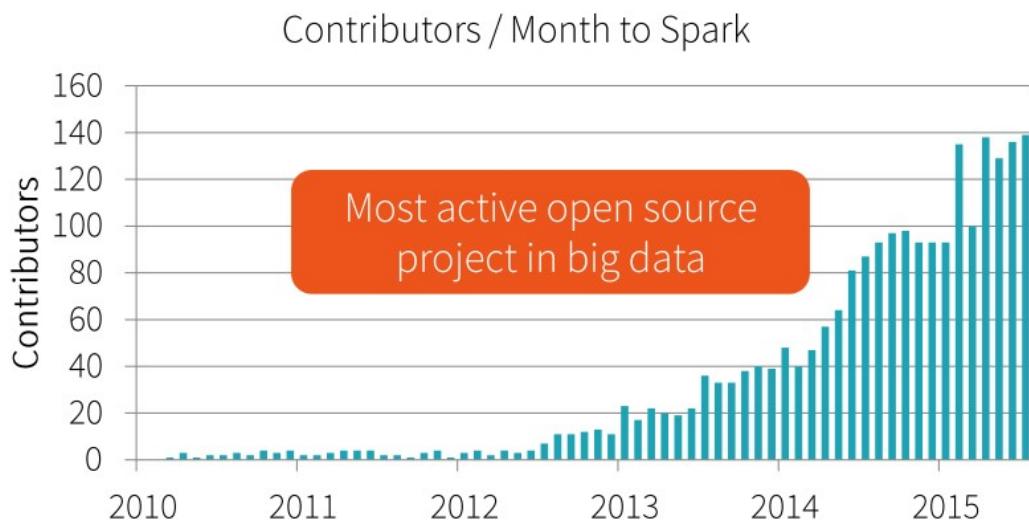
Một trong những lý do khiến Spark chạy nhanh hơn Hadoop MapReduce đó là ở mỗi tác vụ dữ liệu được nạp lên bộ nhớ và xử lý ở đó, những tác vụ sau có thể sử dụng dữ liệu nằm trên bộ nhớ thay vì phải đọc ghi liên tục vào HDFS như Hadoop MapReduce. Theo một so sánh, năm 2013 Hadoop sử dụng cluster bao gồm 2100 máy và mất 72 phút để sắp xếp 100TB dữ liệu, trong khi Spark cần số lượng máy bằng 1/10 nhưng sắp xếp chỉ mất 23 phút. Trong nhiều trường hợp Spark có thể chạy nhanh hơn từ 30 – 50 lần so với Hadoop MapReduce.



Source: Daytona GraySort benchmark, sortbenchmark.org

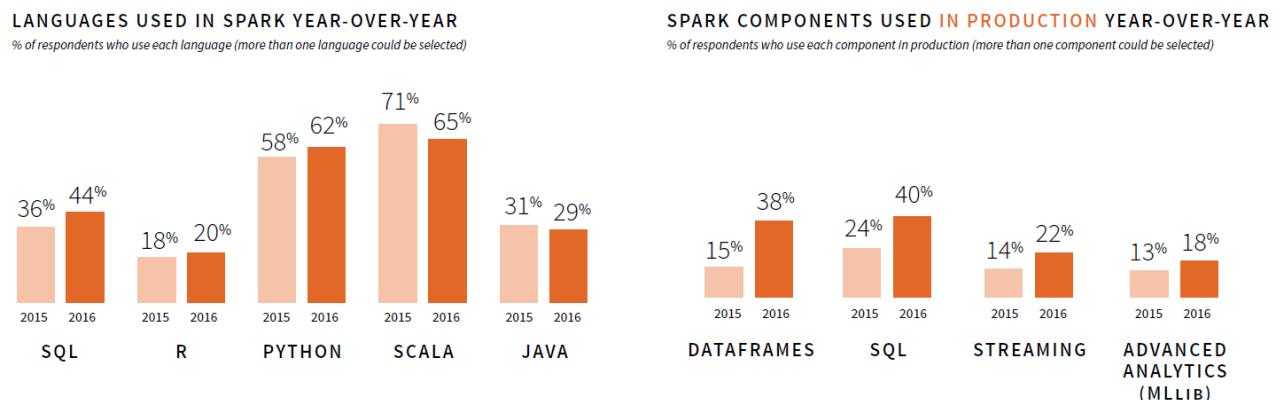
Hình 4.3: So sánh tốc độ sắp xếp 1TB dữ liệu của Apache Hadoop và Apache Spark

Năm 2015, Spark trở thành dự án mã nguồn mở sôi động nhất trong lĩnh vực dữ liệu lớn khi thường xuyên được cập nhật bởi hơn 800 lập trình viên từ 200 công ty trên khắp thế giới.



Hình 4.4: Biểu đồ thể hiện sự quan tâm của cộng đồng đến Apache Spark

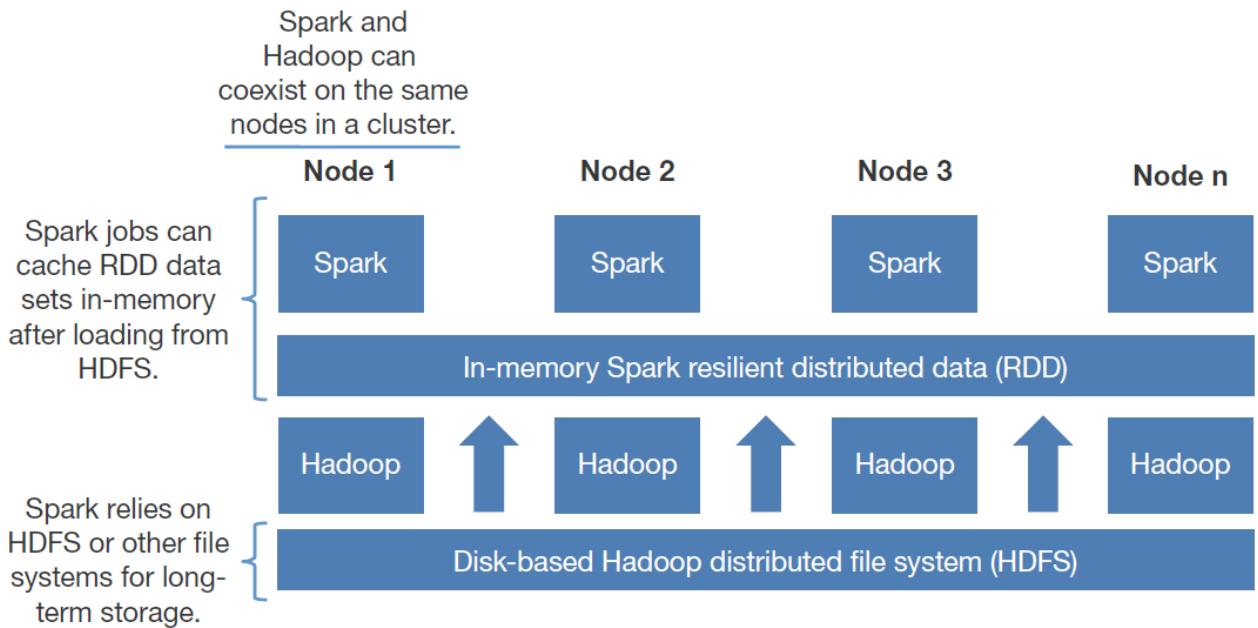
Hình 4.5 cho thấy các ngôn ngữ và hệ sinh thái của spark được sử dụng trong lập trình các ứng dụng. Để thấy Scala và Python là hai ngôn ngữ được sử dụng nhiều nhất trên 58%. Các thành phần trong hệ sinh thái được sử dụng khá đồng đều, được sử dụng nhiều nhất là Dataframes và Spark SQL.



Hình 4.5: Biểu đồ thể hiện sự so sánh mức độ quan tâm của các hệ sinh thái Apache Spark

Trong phần phụ lục A đã trình bày chi tiết cách cài đặt và bảng danh sách các lệnh cơ bản khi làm việc trên hai Apache Hadoop & Spark.

Để tài này sử dụng hệ sinh thái Apache Spark cho bài toán phát hiện cộng đồng trong mạng có kích thước lớn được lưu trữ phân tán trên HDFS của Hadoop như hình 4.6. Cụ thể hơn bài toán sử dụng ngôn ngữ Scala do Apache Spark được xây dựng chủ yếu trên Scala vậy nên có tốc độ và hỗ trợ tốt nhất. Ngoài ra do là bài toán xử lý đồ thị lớn nên Graphx là một lựa chọn tốt.



Hình 4.6: Mô hình kết hợp

4.1.2 Tính toán song song và phân tán với RDD

RDD là từ viết tắt của Resilient Distributed Datasets là một cấu trúc dữ liệu cơ bản của Spark. RDD là một tập hợp bất biến phân tán của một đối tượng. Mỗi dataset trong RDD được chia ra thành nhiều phân vùng logical. Có thể được tính toán trên các node khác nhau của một cụm máy tính (cluster). RDDs API hỗ trợ các ngôn ngữ lập trình như Python, Java, Scala, R trong quá trình phát triển ứng dụng.

Thông thường, RDD chỉ cho phép đọc, phân mục tập hợp của các bản ghi. RDDs có thể được tạo ra trong quá trình xử lý dữ liệu phân tán, RDD là một tập hợp có khả năng chịu lỗi mỗi thành phần có thể được tính toán song song.

Về cơ bản RDD hỗ trợ hai kiểu thao tác thao tác chính:

- **Transformation:** Qua một phương thức transformations thì sẽ cho phép tạo mới một RDD từ RDD đã tồn tại. Tất cả các transformation đều là lazy, có nghĩa là các transformation này sẽ không thực hiện tính toán ngay mà chúng sẽ được lưu lại thành thành các tập lệnh chờ. Quá trình này có thể hiểu như quá trình lập kế hoạch cho một công việc. Nhờ thiết kế này mà Spark chạy hiệu quả hơn.
- **Action:** Sẽ thực hiện lần lượt các thao tác transformations liên quan đến hành động action đã được lên lịch trước và trả về một và một tập giá trị cho chương trình điều khiển.

Bảng 4.1 và 4.2 giới thiệu các thao tác cơ bản thường được sử dụng trong quá lập trình phân tán song song.

Bảng 4.1: Danh sách các thao tác transformation

| STT | Thao tác | Ý nghĩa |
|-----|---|---|
| 1 | map(func) | Trả về 1 RDD mới bằng cách truyền mỗi phần tử đầu vào qua hàm func. |
| 2 | filter(func) | Trả về 1 RDD mới bằng cách chọn những phần tử đầu vào(nguồn) mà hàm func, trả về kết quả true. |
| 3 | flatMap(func) | Tương tự map nhưng khác map ở chỗ, mỗi phần tử đầu vào qua flatMap sẽ trả về 0 hoặc nhiều phần tử đầu ra(có thể hiểu qua map sẽ là 1-1). |
| 4 | mapPartitions(func) | Tương tự như map, nhưng chạy riêng biệt trên mỗi vùng RDD, Hàm func, phải có dạng Iterator[T] => iterator[U] khi chạy RDD kiểu T. |
| 5 | union(otherDataset) | Trả về 1 RDD mới là hợp của tập dữ liệu phần tử đầu vào(nguồn) và các phần tử của đối(otherDataset). |
| 6 | distinct([numTasks]) | Trả về 1 RDD mới chứa mỗi phần tử là duy nhất của tập dữ liệu nguồn. |
| 7 | groupByKey([numTasks]) | Khi gọi đến 1 tập dữ liệu (K,V) sẽ trả về 1 tập là cặp (K, Seq(V)) (Tức là nhóm tập các phần tử cùng Key). Chú ý: mặc định chỉ có 8 task song song khi grouping. Có thể thay đổi số task song song này bằng việc truyền vào tham số đầu vào. |
| 8 | reduceByKey(func, [numTasks]) | Khi gọi tập dữ liệu (K,V), trả về 1 tập (K,V) mà giá trị của key được tổng hợp sử dụng hàm reduce func. |
| 9 | sortByKey([ascending], [numTasks]) | Khi gọi tập dữ liệu (K,V) với K có thể thực hiện sắp thứ tự được. Khi đó, nó sẽ trả về tập dữ liệu (K,V) được sắp xếp tăng dần hoặc giảm dần theo key. Chú ý: ascending là kiểu Boolean. |
| 10 | join(otherDataset, [numTasks]) | Khi gọi tập dữ liệu có kiểu (K,V) và (K,W), nó sẽ trả về 1 cặp mới (K,(V,W)) (nối 2 phần tử có cùng key). |
| 11 | cogroup(otherDataset, [numTasks]) | Khi gọi tập dữ liệu có kiểu (K,V) và (K,W), nó sẽ trả về 1 tập dữ liệu (K, seq(V), seq(W)). |
| 12 | cartesian(otherDataset) | Khi gọi 1 tập dữ liệu kiểu T và U, nó sẽ trả về tập dữ liệu mới (T,U). |

Bảng 4.2: Danh sách các thao tác action

| STT | Thao tác | Ý nghĩa |
|-----|-----------------------------|--|
| 1 | reduce(func) | Tổng hợp các phần tử của tập dữ liệu sử dụng hàm func(có 2 đối và trả về 1 kết quả). |
| 2 | collect() | Trả về tất cả các phần tử của tập dữ liệu như 1 mảng ở driverProgram. Hàm này hữu ích sau khi lọc hoặc thao tác khác mà trả về tập dữ liệu con đủ nhỏ. |
| 3 | count() | Trả về số phần tử của tập dữ liệu |
| 4 | first() | Trả về phần tử đầu tiên của tập dữ liệu(tương tự take(1)). |
| 5 | take(n) | Trả về mảng gồm n phần tử đầu tiên của tập dữ liệu. |
| 6 | saveAsTextFile(path) | Ghi các phần tử của tập dữ liệu như 1 file text(hoặc tập file text) lên 1 thư mục trong hệ thống local, HDFS hoặc hệ thống hỗ trợ Hadoop bất kỳ. |
| 7 | countByKey() | Chỉ cho RDD có kiểu (K,V). Trả về 1 Map (K,Int). Int là chỉ số key. |
| 8 | foreach(func) | Chạy hàm func cho mỗi phần tử của tập dữ liệu. Điều này có tác dụng khi thực hiện cập nhật 1 biến accumulator hoặc tương tác với hệ thống lưu trữ ngoài. |

4.2 Thực nghiệm

Để kiểm chứng phương pháp phát hiện cộng đồng sử dụng mô hình BigCLAM, phương pháp đã được tiến hành cài đặt thực nghiệm trên môi trường cụm máy tính có cấu hình như bảng 4.3 sử dụng Apache Hadoop trong việc lưu trữ phân tán và Apache Spark trong tính toán phân tán. Trong quá trình thực nghiệm, có sử dụng một vài cấu trúc mạng có kích thước tương đối lớn như trong bảng 4.4.

Bảng 4.3: Thông số hệ thống sử dụng trong thực nghiệm

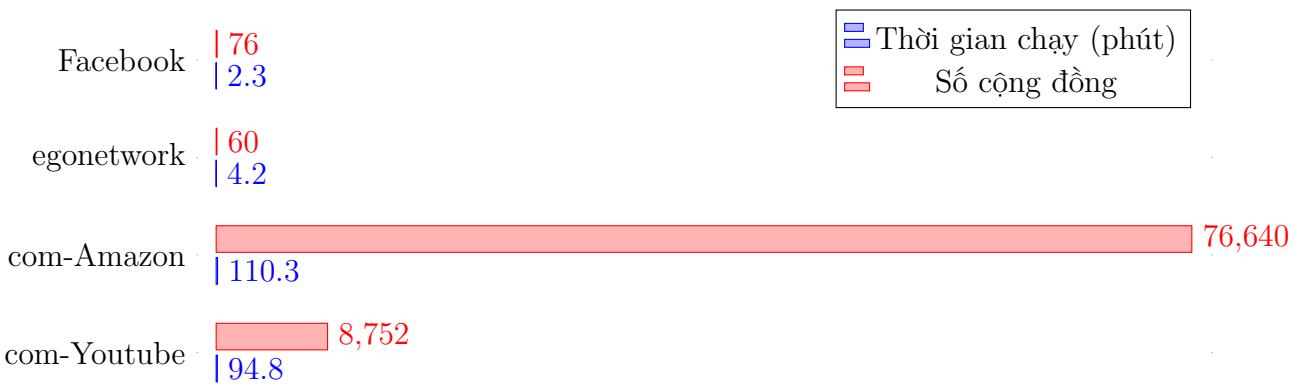
| STT | Thông số | Số lượng | Vai trò |
|-----|---|----------|---------|
| 1 | OS: Ubuntu 16.04 LTS HDD: 95 GB RAM: 4 GB CPU: intel Core i5-4590U CPU 3.30GHz x 4 | 1 | master |
| 2 | OS: Ubuntu 16.04 LTS HDD: 95 GB RAM: 4 GB CPU: intel Core i5-4590U CPU 3.30GHz x 4 | 9 | workers |

Trong quá trình thực nghiệm tôi đã sử dụng sử dụng các thao tác với RDDs như trong bảng 4.1 và 4.2 để thực hiện tính toán phân tán.

Bảng 4.4: Mô tả các mạng sử dụng để thực nghiệm

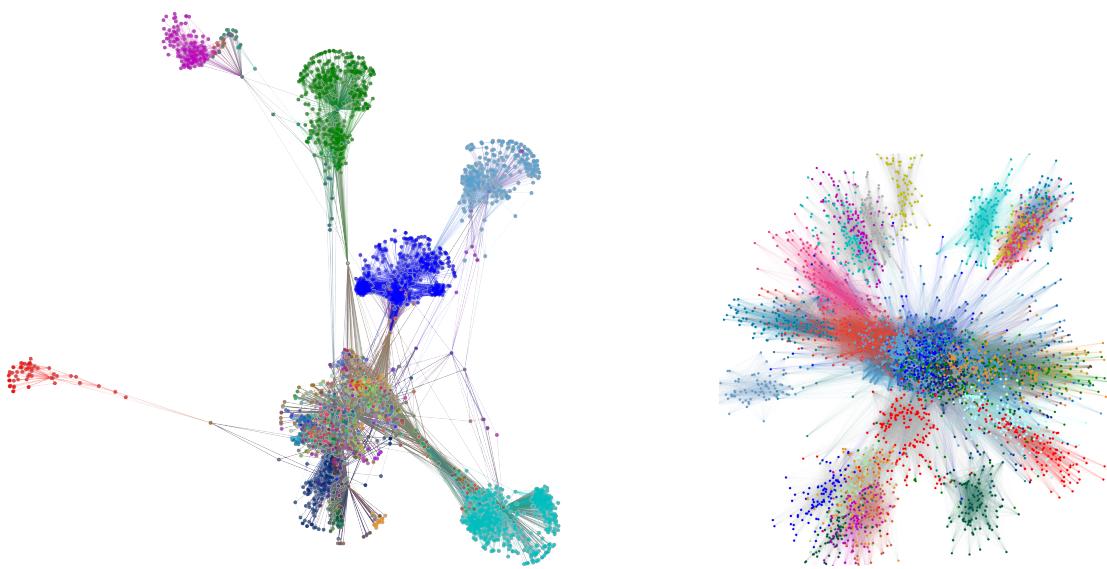
| Tên mạng | Số đỉnh | Số cạnh |
|----------------------|-----------|-----------|
| Mạng xã hội facebook | 4,039 | 88,234 |
| Mạng egonetwork | 20,812 | 27,282 |
| Mạng com-Youtube | 1,134,890 | 2,987,624 |
| Mạng com-Amazon | 334,863 | 925,872 |

Biểu đồ thể hiện thời gian chạy và số lượng cộng đồng phát hiện được trên mỗi mạng



Hình 4.7: Kết quả thực nghiệm

Hình 4.7 chính là kết quả quá trình thực nghiệm của các mạng trong bảng 4.4. Có thể thấy



Hình 4.8: Mô phỏng phát hiện cộng đồng trên mạng facebook 4.4
 Hình 4.9: Mô phỏng phát hiện cộng đồng trên mạng internet 4.4

thời gian thực nghiệm trên khá tốt. Và hình 4.8 và 4.9 là kết quả của việc mô phỏng phân bố của các cộng đồng trong mạng xã hội facebook và Internet trong bảng 4.4.

Ngoài ra, việc sử dụng một vài khái niệm sau đã cải thiện hiệu năng đáng kể:

- Biến broadcast: được tạo ra bằng cách gọi hàm `SparkContext.broadcast(v)`. Broadcast variable là một wrapper xung quanh biến `v`, giá trị của biến này có thể được truy xuất thông qua hàm `value`. Về cơ bản biến broadcast sẽ được phân tán biến biến local trên tất cả các máy worker, điều này làm giảm thời gian cho các máy worker quay trở lại máy master để lấy giá trị.
- Biến accumulator: Khi ta muốn thực hiện các thao tác trên tập dữ liệu được phân tán mà không thể thông qua các thao tác RDD hỗ trợ thì biến accumulator cho phép là một biến trung gian (tổng) giữa các máy trong cụm.
- phương thức cache/ persist: Cho phép dữ liệu được lưu trữ trên RAM. Điều này giúp cho các tác vụ được sử dụng nhiều lần sẽ có thời gian truy vấn nhanh hơn do dữ liệu đã lưu lại trên RAM.

Để quá trình huấn luyện theo công thức 3.5 trên hệ thống phân phát huy tính hiệu quả. Tôi đã đề xuất sử dụng phương pháp tăng gradient ngẫu nhiên theo cụm đã được trình bày trong thuật toán 4.

Chương 5

Kết luận và hướng phát triển

5.1 Kết luận

Khóa luận này tập trung làm rõ bài toán phát hiện cộng đồng, đặc biệt với những cộng đồng có tính chồng chéo, lồng nhau. Đồng thời nhấn mạnh mức độ quan trọng của bài toán trong thực tế hiện nay, khi mà các mạng ngày càng trở lên mở rộng cả về cấu trúc lẫn nội dung có thể thấy một ví dụ điển hình là mạng xã hội Facebook với dân số gần 2 tỷ người được ví như là một quốc gia ảo.

Nội dung chính của khóa luận là làm rõ mô hình BigCLAM, từ đó đề xuất sử dụng một vài phương pháp tối ưu hàm lồi trong phần 2.3.2 cũng như phương pháp khởi tạo matrix trọng số giúp cho quá trình phát hiện hiệu quả và nhanh hơn.

Đặc biệt, phần chính của khóa luận này hướng đến là cài đặt thực nghiệm phương pháp phát hiện cộng đồng trên cụm máy tính sử dụng hai framework Apache Hadoop trong việc lưu trữ phân tán và Apache Spark trong quá trình tính toán phân tán. Việc chuyển từ bài toán chạy trên một máy sang bài toán phân tán ra nhiều máy là một bài toán không hề dễ dàng nhưng hiệu quả của nó mang lại rất lớn khi mà dữ liệu ngày một lớn và không ngừng tăng trưởng. Lợi thế của hệ thống phân tán là có thể mở rộng tính toán theo chiều ngang vậy nên tôi đã đề xuất phương pháp giảm gradient ngẫu nhiên theo cụm (MBSGD) được trình bày trong thuật toán 4. Tuy nhiên, trong quá trình thực nghiệm những dữ liệu lớn hơn khoảng vài triệu điểm và cặp cạnh lại cho thấy thời gian khá chậm với nhu cầu thực tế. Điều này cần phải được nghiên cứu kỹ hơn cả về kiến trúc hạ tầng lẫn tối ưu quá trình lập trình phân tán.

Kết luận, khóa luận về cơ bản đã hoàn thành mục tiêu đã đề ra từ đầu trong việc giải quyết bài toán phát hiện cộng đồng trọng lượng tương tác có kích thước lớn. Tuy nhiên, để áp dụng thuật toán vào thực tế cần phải được đầu tư nghiêm túc để phát huy sức mạnh của thuật toán và hệ thống phân tán.

5.2 Hướng phát triển

Từ những kết luận trên, trong thời gian tới, tôi sẽ tiếp tục nghiên cứu các phương pháp tối ưu và kỹ thuật lập trình phân tán song song trên Apache Spark/ Hadoop để cải tiến tốc độ bài toán phù hợp với nhu cầu phân tích thời gian thực trên các mạng hiện nay. Đồng thời đưa phương pháp này giải quyết các bài toán trong thực tế. Ví dụ bài toán xây dựng hệ khuyến nghị trong mạng xã hội giáo dục Đại học Thăng Long đề tài được đang được tiến hành phát triển bởi một nhóm sinh viên Đại học Thăng Long.

Tài liệu tham khảo

- [1] Yong-Yeol Ahn, James P Bagrow, and Sune Lehmann. Link communities reveal multiscale complexity in networks. *Nature*, 466(7307):761–764, 2010. [20](#)
- [2] Edoardo M Airoldi, David M Blei, Stephen E Fienberg, and Eric P Xing. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research*, 9(Sep):1981–2014, 2008. [20](#)
- [3] John Aldrich et al. Ra fisher and the making of maximum likelihood 1912-1922. *Statistical Science*, 12(3):162–176, 1997. [11](#)
- [4] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT’2010*, pages 177–186. Springer, 2010. [14](#)
- [5] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004. [12](#), [13](#)
- [6] David F. Gleich and C. Seshadhri. Neighborhoods are good communities. *CoRR*, abs/1112.0031, 2011. [10](#), [19](#)
- [7] Steve Gregory. An algorithm to find overlapping community structure in networks. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 91–102. Springer, 2007. [6](#)
- [8] Steve Gregory. Fuzzy overlapping communities in networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2011(02):P02017, 2011. [21](#)
- [9] Steve Gregory. Fuzzy overlapping communities in networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2011(02):P02017, 2011. [21](#)
- [10] Jakub Konečný, Jie Liu, Peter Richtárik, and Martin Takáč. Mini-batch semi-stochastic gradient descent in the proximal setting. *IEEE Journal of Selected Topics in Signal Processing*, 10(2):242–255, 2016. [15](#)

- [11] David G Luenberger. *Introduction to linear and nonlinear programming*, volume 28. Addison-Wesley Reading, MA, 1973. [14](#)
- [12] Julian J. McAuley and Jure Leskovec. Discovering social circles in ego networks. *CoRR*, abs/1210.8182, 2012. [4](#)
- [13] Mark EJ Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69(6):066133, 2004. [5](#)
- [14] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, 2005. [20](#)
- [15] Jaewon Yang and Jure Leskovec. Community-affiliation graph model for overlapping network community detection. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 1170–1175. IEEE, 2012. [7](#), [16](#)
- [16] Jaewon Yang and Jure Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 587–596. ACM, 2013. [5](#), [7](#), [16](#)

Phụ lục A:

Cài đặt và triển khai ứng dụng trên Apache Hadoop & Spark

Tại chương 4 triển khai phương pháp phát hiện cộng đồng được trình bày ở chương 3 trên hai framework Apache Hadoop và Apache Spark. Trong chương này, tôi sẽ trình bày các cài đặt và triển khai ứng dụng trên hai framework này.

A.1 Cài đặt

Apache Hadoop và Spark được phát triển trên nền tảng GNU/Linux và hoạt động tốt trên môi trường này. Vì vậy, chúng ta nên sử dụng hệ điều hành Linux để cài đặt hai framework này. Giả sử ta sẽ cài đặt hai framework trên một cụm máy gồm 3 node (worker) và 1 node (master).

Dầu tiên, ta cần xác định địa chỉ mạng và thiết đặt địa chỉ tĩnh trên mỗi máy trong cụm. Điều này nhằm cố định địa chỉ mạng trên mỗi máy tính. Ví dụ như sau:

Bảng 1: Các địa chỉ IP trong cụm

| STT | Địa chỉ IP | Vai trò |
|-----|----------------|---------|
| 1 | 192.168.100.10 | master |
| 2 | 192.168.100.11 | worker1 |
| 3 | 192.168.100.12 | worker2 |
| 4 | 192.168.100.13 | worker3 |

Bước tiếp theo, ta tiến hành cập nhật file hosts (*/etc/hosts*) theo bảng 1 trên mỗi máy tính. Mục đích là để 4 node này nó thấy nhau qua tên máy hoặc IP.

Để xác thực giữa các máy trong cụm với nhau nhằm tạo một cơ chế bảo mật và không yêu cầu mật khẩu qua mỗi lần khởi chạy, ta sẽ sử dụng thông qua cơ chế SSH. Để cài đặt SSH ta dùng lệnh:

```
$ sudo apt-get install openssh-server openssh-client
```

Sau đó tiến hành sinh khóa RSA và sao chép khóa publickey cho các máy *worker* (thực hiện trên máy *master*):

```
$ ssh-keygen  
$ cd ~/.ssh  
$ ssh-copy-id -i id_rsa.pub sv@worker1  
$ ssh-copy-id -i id_rsa.pub sv@worker2  
$ ssh-copy-id -i id_rsa.pub sv@worker3  
$ cat id_rsa.pub >> authorized_keys
```

Sửa tên máy cho đúng với *master* và từng *worker*, chạy lệnh:

```
$ hostnamectl set-hostname master  
$ hostnamectl set-hostname worker1  
$ hostnamectl set-hostname worker2  
$ hostnamectl set-hostname worker3
```

A.1.1 Apache Hadoop & Spark

Tiến hành tải và cài đặt các gói cài đặt Java, Scala, Apache Hadoop, Apache Spark (Chú ý chọn các phiên bản java, scala hỗ trợ hadoop và spark). e

- **Scala:** <http://www.scala-lang.org/download/>
- **Java:** <https://www.java.com/en/download/>
- **Apache Hadoop:** <http://hadoop.apache.org/releases.html>
- **Apache Spark:** <http://spark.apache.org/downloads.html>

Tại thời điểm thực hiện đề tài, tôi sử dụng các phiên bản sau:

Bảng 2: Các phiên bản cài đặt

| STT | Gói cài đặt | Phiên bản |
|-----|---------------|-------------------------------|
| 1 | Scala | scala-2.12.1.tgz |
| 2 | Java | jdk-8u121-linux-x64.tar.gz |
| 3 | Apache Hadoop | hadoop-2.7.2.tar.gz |
| 4 | Apache Spark | spark-2.1.0-bin-hadoop2.7.tgz |

Tiến hành giải nén và chuyển vào mục cài đặt các gói cài đặt:

```
// Giải nén gói cài đặt
$ tar -xvf jdk-8u121-linux-x64.tar.gz
$ tar -xvf hadoop-2.7.2.tar.gz
$ tar -xvf spark-2.1.0-bin-hadoop2.7.tgz
$ tar -xvf scala-2.12.1.tgz
// Tạo thư mục cài đặt
$ sudo mkdir /usr/local/java
$ sudo mkdir /usr/local/hadoop
$ sudo mkdir /usr/local/spark
$ sudo mkdir /usr/local/scala
// Chuyển file cài đặt vào thư mục cài đặt
$ sudo mv jdk1.8.0_121 /usr/local/java/jdk1.8.0_121
$ sudo mv hadoop-2.7.2 /usr/local/hadoop/2.7.2
$ sudo mv spark-2.1.0-bin-hadoop2.7
/usr/local/spark/2.1.0
$ sudo mv scala-2.12.1 /usr/local/scala/scala-2.12.1
```

Sau đó ta tiến hành thiết lập (*nano ~/.bashrc*) và cập nhật môi trường(*source ~/.bashrc*) như sau:

```
export JAVA_HOME=/usr/local/java/jdk1.8.0_121
export JRE_HOME=$JAVA_HOME/jre
export SCALA_HOME=/usr/local/src/scala/scala-2.12.1
export SPARK_HOME=/usr/local/spark/2.1.0
export HADOOP_HOME=/usr/local/hadoop/2.7.2
export PATH=$PATH:$JAVA_HOME/bin:$JRE_HOME/bin:$SCALA_HOME/bin:
```

```
$SPARK_HOME/bin:$SPARK_HOME/sbin:  
$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

A.1.2 Cấu hình Hadoop

Cấu hình danh sách các máy làm *master*:

```
$ nano $HADOOP_HOME/etc/hadoop/masters
```

Cấu hình danh sách các máy làm *worker*:

```
$ nano $HADOOP_HOME/etc/hadoop/slaves
```

Khởi tạo namenode cho máy master

```
$ mkdir -p $HADOOP_HOME/hadoop_store/hdfs/namenode
```

Khởi tạo datanode cho máy worker

```
$ mkdir -p $HADOOP_HOME/hadoop_store/hdfs/datanode  
$ chmod 755 $HADOOP_HOME/hadoop_store/hdfs/datanode
```

Cấu hình core-site.xml:

```
<property>  
<name>fs.defaultFS</name>  
<value>hdfs://master:8020</value>  
</property>
```

Cấu hình hdfs-site.xml:

```
<property>  
<name>dfs.namenode.name.dir</name>  
<value>file:/usr/local/hadoop/2.7.2/hadoop_store/hdfs/namenode</value>  
</property>  
<property>  
<name>dfs.datanode.data.dir</name>  
<value>file:/usr/local/hadoop/2.7.2/hadoop_store/hdfs/datanode</value>  
</property>  
<property>  
<name>dfs.replication</name>  
<value>3</value>  
</property>
```

Cấu hình mapred-site.xml:

```
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
```

Cấu hình yarn-site.xml:

```
<property>
<name>yarn.resourcemanager.resource-tracker.address</name>
<value>master:8025</value>
</property>
<property>
<name>yarn.resourcemanager.scheduler.address</name>
<value>master:8030</value>
</property>
<property>
<name>yarn.resourcemanager.address</name>
<value>master:8050</value>
</property>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
<value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
<property>
<name>yarn.nodemanager.disk-health-checker.min-healthy-disks</name>
<value>0</value>
</property>
```

Tiến hành định dạng HDFS:

```
$ $HADOOP_HOME/bin/hdfs namenode -format
```

A.1.3 Cấu hình Spark

Cấu hình danh sách các máy làm *worker*:

```
$ nano $SPARK_HOME/conf/slaves
```

Cấu hình spark-env.sh:

```
SPARK_WORKER_MEMORY=1g  
SPARK_WORKER_INSTANCES=1  
SPARK_WORKER_CORES=2  
SPARK_MASTER_HOST=master  
HADOOP_CONF_DIR=/usr/local/hadoop/2.7.2/etc/hadoop
```

Cấu hình spark-defaults.sh:

```
spark.master spark://192.168.100.10:7077
```

A.2 Tập lệnh

A.2.1 Danh sách tập lệnh làm việc trên Hadoop

Bảng 3: Danh sách tập lệnh cơ bản làm việc trên Hadoop

| STT | Lệnh | Ý nghĩa |
|-----|--|-------------------------------------|
| 1 | hadoop namenode -format | format hdfs |
| 2 | \$HADOOP_HOME/sbin/start-dfs.sh | Khởi động hdfs: http://master:50070 |
| 3 | \$HADOOP_HOME/sbin/start-yarn.sh | Khởi động yarn: http://master:8088 |
| 4 | \$HADOOP_HOME/sbin/stop-dfs.sh | Dừng hdfs |
| 5 | \$HADOOP_HOME/sbin/stop-yarn.sh | Dừng yarn |
| 6 | \$HADOOP_HOME/sbin/start-all.sh | Khởi động hdfs + yarn |
| 7 | \$HADOOP_HOME/sbin/stop-all.sh | Dừng hdfs + yarn |
| 8 | \$HADOOP_HOME/sbin/hadoop-daemon.sh start datanode | Khởi động trên máy worker |
| 9 | hadoop fs -mkdir -p /Documents/Data | Tạo folder Data trên HDFS |
| 10 | hadoop fs -rm /Documents/Data/datatest.txt | Xóa file Data trên HDFS |
| 11 | hadoop fs -rmdir /user/output | Xóa folder Data trên HDFS |
| 12 | hadoop fs -ls /user/output | Liệt kê file trên HDFS |
| 13 | hadoop fs -cat /user/output/NTMerged.txt | Hiển thị nội dung file |
| 14 | hadoop fs -cat data.txt /Document/Data/ | Đẩy file local lên HDFS |

A.2.2 Danh sách tập lệnh làm việc trên Spark

Bảng 4: Danh sách tập lệnh cơ bản làm việc trên Spark

| STT | Lệnh | Ý nghĩa |
|-----|--------------------------------|--|
| 1 | \$SPARK_HOME/sbin/start-all.sh | Khởi động Spark |
| 2 | \$SPARK_HOME/sbin/stop-all.sh | Dừng Spark |
| 3 | \$SPARK_HOME/bin/spark-shell | Lập trình ứng dụng sử dụng ngôn ngữ scala |
| 4 | \$SPARK_HOME/bin/pyspark | Lập trình ứng dụng sử dụng ngôn ngữ python |
| 5 | \$SPARK_HOME/bin/sparkR | Lập trình ứng dụng sử dụng ngôn ngữ R |
| 6 | \$SPARK_HOME/bin/spark-submit | Tạo ứng dụng |